Latent Variable Methods for Visualization Through Time

Iain Guy David Strachan

Doctor of Philosophy

Institute for Adaptive and Neural Computation
Division of Informatics
University of Edinburgh

2002

Abstract

This doctoral thesis presents the results of my work into the visualization of high dimensional time-dependent data. Specifically, two new algorithms are presented: Probabilistic Principal Components Through Time, and GTM Through Time. The former is a linear mapping taking time dependence into account, which is a special case of the linear Kalman Filter. The latter is a non-linear mapping technique where a non-linear two-dimensional manifold, discretized as a regular grid of points, is fitted through the data. Each grid point corresponds to a hidden state of a Hidden Markov Model. The model builds on the existing Generative Topographical Mapping (GTM) technique for time-independent data. Both techniques utilize a particular form of probabilistic graphical model, namely a Directed Acyclic Graph, in order to achieve the extension from a time-independent model to the time dependent case. The models are trained by maximizing the log likelihood of the data given the model using the Expectation–Maximization (EM) algorithm. The behaviour of the techniques is demonstrated using toy data, and results are also presented using a real engineering data set, from a human patient monitoring application.

Acknowledgements

I should like first of all to thank my supervisor Prof. Chris Bishop, for encouraging me to undertake this research, and for having faith in my abilities, based on our previous work together at AEA Technology. Next, thanks are due to AEA Technology, my employer, for sponsoring me for the work. Particular thanks are due to my colleagues Lawrence Daniels and Bill Worraker, for many stimulating and helpful discussions, which have helped me to formulate and express the ideas in this thesis.

I would also like to acknowledge the help of Prof. Lionel Tarassenko for his encouragement and support during this period of study, and for suggesting that I apply for a Wolfson-Harwell industrial fellowship, that allowed me to work in an academic environment for one day a week during a three year period. It has been of enormous benefit to be able to interact with the group at Oxford, which also provided the data for study in the applications chapter of the thesis. I also am grateful to the President and Fellows of Wolfson College for electing me to the fellowship and enabling this collaboration to take place.

I should also like to express my gratitude to Diana Calvert for proof-reading the thesis. Her professional experience in proof reading scientific publications has helped enormously to improve the consistency of terminology used throughout the thesis.

Last, by not least, I should like to thank my loving wife, Christine, and my children Jessica and Matthew for being so supportive, and for patiently enduring my long absences from family affairs while working on the thesis.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Iain Guy David Strachan*)

To the memory of my father, Norman Mervyn Strachan (1921–1993).

Table of Contents

Chapter 1

Introduction

André Previn: *You're playing all the wrong notes!*
Eric Morecambe: *I'm playing all the right notes, but not necessarily in the right order.*
("The Morecambe and Wise Christmas Show", 1968)

This thesis is concerned with processing, visualization and interpretation of ordered sequences of data. What we aim to achieve is to develop automated techniques which produce a visualization of such data, taking into account the temporal context, as opposed to treating each observation independently of all the others (we say that such data points are *independent and identically distributed*, or i.i.d.). We shall specifically develop probabilistic time-dependent visualization algorithms that extend existing algorithms in use for i.i.d. data. These existing algorithms are based on a latent variable approach, which can be represented in terms of probabilistic graphical models. This formulation leads to a natural way of extending to the time-dependent case.

The interpretation of time-dependent data, taking into account the context, rather than treating each sample as independent of all the others, is akin to the interpretation of a piece of music by a skilled musician. A musical score may be regarded as a highly stylized form of data visualization for time dependent data. Specifically, it is a kind of multiple graph of pitch against time for a number of different sources. While some musical scores contain much extra information to assist the player (such as how loud or soft to play a note, how fast to play a passage, written instructions pertaining to

style and mood, and so forth), some scores contain little more than the raw information of a sequence of notes.

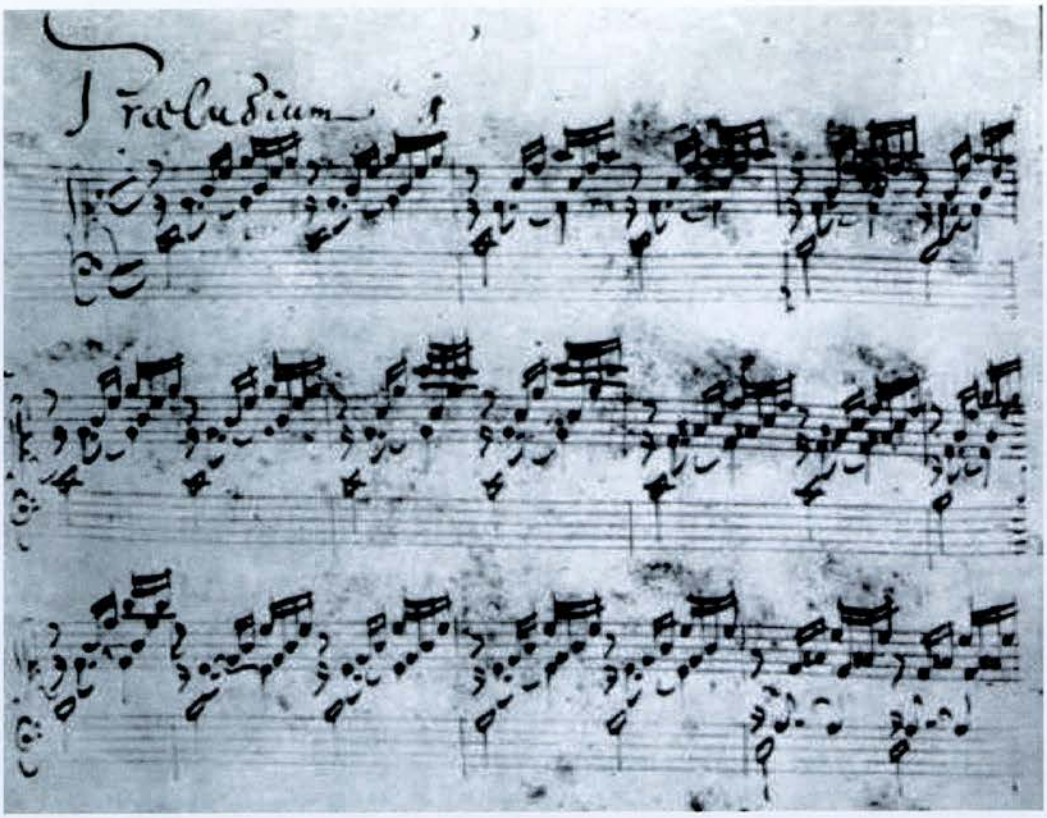


Figure 1.1: Manuscript of the "C major Prelude" from "The Well-Tempered Klavier", Book 1, by J.S. Bach.

Such a score is illustrated in Figure 1.1, and the processes involved whereby a musician comes to interpret and produce a meaningful performance of such a piece of music will serve as a non-technical illustration of the motivating concerns behind this study. The figure shows the original manuscript, in the composer's hand, of the famous "Prelude in C major" from Book 1 of "The Well-Tempered Klavier" by Johann Sebastian Bach (1685–1750)[1]. In common with many of Bach's manuscripts, it contains little clue as to how

[1]Historical footnote. It will appear to a modern musician familiar with this piece, that the upper staff contains "all the wrong notes". This is because the familiar treble clef of modern scores was not adopted until the end of the 18th Century, [Sadie, 2001, Vol 6, p 24]. In this score, an archaic form of C-clef is used, placed to indicate that the bottom line of the staff corresponds to middle C, which means all the notes are one line higher than in modern scores. The placement of the C-clef on the first line is also no longer in use; nowadays it is only placed on the third or fourth line in the staff.

to perform the piece; there is no indication of dynamic level or phrasing. It is simply a graph of notes, that could just as easily be reduced, without loss of information, into a file of numbers. How does one interpret such data? A beginner musician, presented with such a score will primarily concentrate on simple things, such as playing all the right notes, and in the right order. However once that has been mastered, and all the information that is present in the score has been learnt, there is a great deal more to be accomplished before a meaningful interpretation can be produced.

Listening to this particular piece of music produces a sense of inexorability — the feeling that each successive bar proceeds in a logical fashion from the last, and that the whole piece is a continuous journey, from beginning to end, that was deliberately fashioned that way. It follows that if one is to produce that sense of direction in a performance (or "aural visualization") of the work, that one must also be conversant with the whole piece, and be aware at any point of where we are in the piece.

While it is not the intention here to reduce the analysis of music to probabilistic inference, nonetheless probabilistic considerations turn out to provide some limited insight into this piece. This leads to some useful analogies with what we are attempting to achieve in this study.

Firstly we note that even a non-time dependent analysis of the music might reveal certain useful information. Music written in Bach's time (and up to the 20th Century) was **tonal**, meaning that it was rooted to a particular key. One of the implications of this is that we expect the probabilities of occurrence of the seven notes in the scale will tend to be higher than those of the other five. In the key of C major, we thus expect the "white" notes of the keyboard to occur more frequently than the "black" notes, because the scale of C major consists of the white notes beginning at C. Additionally certain important notes in the scale (C, E and G) may well be more frequent — certainly at the start of the piece — than the others. We can formulate this as a prior distribution over the 12 possible pitches in the octave. In the C major prelude, in fact all 12 pitches are used at one time or another, but less than 10 percent of the notes are "black notes" because these are not part of the scale. Such notes are technically termed "accidentals", and the very word

has probabilistic connotations, implying that they are relatively rare events[2]. The occurrence of relatively rare events can then assist in interpration; and a performer may choose to emphasize these events in a particular way.

However it is not simply the prior distribution of note pitches that determines the interpretation; it is also the *sequence* of notes, and so the problem is a time-dependent one. Examination of the manuscript of the music shows that there is a repetitive figure with the same basic shape throughout the piece, and the figure is based around five notes. Each of these is known as a "broken chord", and the progression of chords through time would be used to assist in the interpretation of the music, in that not only would unusual *notes* be emphasized (as in a static analysis), but also unusual *sequences* of chords, even if the actual notes are not unusual. Less likely sequences would be indicative of changes taking place in the musical narrative, and would be highlighted by the performer. In this particular piece, the fourth bar of music is identical to the first, and the fifth bar sets off in a new direction, and is followed by the first incidence of an accidental note (an F–sharp) in the sixth bar. Knowledge of the temporal context allows us to pick up that something new is happening earlier than the static analysis, which only detects an unusual event with the appearance of the accidental.

One of the ways in which music might be said to work is via a predictive model that operates in the listener's mind. At any time, we have an expectation of what will happen next, and the music alternately satisfies and contradicts the expectations. It is important that the predictive model is good enough so that for the most part it is able to predict accurately (otherwise the music will seem an incomprehensible sequence of notes), but also important that it does not predict accurately every time, or else the music will be devoid of interest.

Equally, in using probabilistic models to interpret real data, it is of use to be able to detect unusual events, and to flag these up as unusual, or requiring attention. This will happen when the model prediction shows poor agreement with the data, and then some kind of alarm can be raised. And by analogy

[2]By contrast, in the music of "atonal" composers, such as Arnold Schoenberg (1874-1951), we should expect there to be a uniform prior over the 12 pitches in the scale, and that therefore these simple ideas cannot be applied to interpretation.

with the idea of an incomprehensible sequence of notes, a model that shows poor agreement all of the time will generate too many false alarms.

In the example of the Bach prelude, there is no distinction between the data and what is predicted by the model. In this thesis, we shall use the term *Latent Variable*, which is a variable that is not observed, but which governs the underlying processes taking part. The latent variable model is formulated as a probability distribution in high dimensional data space, but whose independent variables are in a low dimensional space, known as *latent space.*

There are two different ways in which a latent variable model formulated in this way can be used. Firstly, we can obtain a visualization of a high-dimensional data set by plotting the latent variables (which are assumed to be of low dimension). Secondly, we can take a sequence of latent variables, and use it to *generate* "fantasy data", by sampling the probability distribution with the aid of a random number generator.

In terms of music, this could be understood in terms of the relationship between melody and harmony. The Bach Prelude is unusual in being an interesting sequence of harmonies, apparently with no melody overlaid, but later on the composer Gounod (1818-1893) used it as the basis for an arrangement of the "Ave Maria", where the sequence of notes from the Prelude was combined with a melody to which the words were sung. The melody would have been composed by choosing a sequence of notes each of which combined naturally with the harmonies underneath (the five note sequences). When we listen to a song, what we remember is the tune, which constitutes the "observed data". The harmonies beneath it then take on a secondary rôle, and might be considered as a kind of latent variable, whose presence enhances our appreciation of the music. In the case of Gounod's "Ave Maria", the underlying sequence was pre-specified, and used generatively to produce the tune.

Naturally, we are not suggesting here that Gounod used a random process to generate the melody; however the presence of each successive chord, defined by the five note sequences, had a strong influence on the notes of the melody, restricting the choice to a set of notes which combined well with the it, some of which were more likely to be chosen than others. One could in

principle construct a generative model with the Bach prelude as the underlying sequence, and probabilistic rules for selecting the melody notes. Given such a model, we could then compute the likelihood of Gounod's melody from it.

However, it is also quite feasible to think up a melody, and then to find the correct harmonies to accompany the tune[3]. It is this kind of process that we consider mainly in this thesis; our latent, visualization variables are inferred from the raw data, in order to assist in our understanding of the temporal evolution of the data, i.e. deducing the harmony from the melody.

Many of the current techniques for data visualization (such as Principal Components Analysis, or the Kohonen Self-Organizing Map), do not exploit the time dependence of the data; the results presented would be the same, even if the data vectors were processed in a randomized order. They are therefore akin to the static analysis of music where we simply count the relative frequencies of occurrence of the different note pitches. Such a technique would not pick up anything abnormal about Eric Morecambe's playing referred to in the quotation at the beginning of the thesis. A time-dependent visualization technique should be able to give different results with data that is presented in the wrong order, and flag the sequence as abnormal, by assigning a low probability to the sequence, or by producing a recognizably different visualization.

1.1 Data Visualization

1.1.1 Visualization and Information Preservation

In data visualization applications, there will nearly always be a loss of information, because the latent space is of lower dimension than the data space, and we do not expect there to be an exact fit. What we are concerned with is that the information thrown away is irrelevant (such as extraneous

[3]There is some debate over the artistic validity of the use of Bach's Prelude to generate a new melody like this. Many would argue that there are melodies "hidden" within the chord sequence of the prelude, but this does not fit well with the illustration here, that the chord sequence is the underlying process that generates the melody, and is analogous to the latent variable.

noise), thereby leaving a clearer view of the relevant information. Usually data records have many different fields, and as such are represented as a high dimensional vector of numbers. In order to produce a visualization, we must reduce the dimensionality of the visualization space to two or possibly three. This clearly involves throwing away information, and leads to the possibility that mistakes can be made. A new example can *appear* to be in the middle of a cluster in the visualization space, but at the same time be nowhere near it in data space, which may lead to misinterpretation. Thus an important characteristic of data visualization techniques is the preservation of information. Trivially, one can see that a visualization that plots two observed variables on a graph or scatter plot throws away all the information from the other variables. More sophisticated linear projection techniques attempt to find a projection from D-dimensional space onto the two-dimensional visualization space in such a way as to reflect some of the information contained in all the variables. Non-linear mapping techniques preserve more information by considering projections onto a non-linear surface.

1.1.2 Visualization and Feature Selection

The extraction of feature vectors is, of course, not restricted to visualization techniques. Some of these techniques may be applied to feature selection for neural networks, in which case the dimension of the feature vector is not restricted to be a low value.

For example the technique known as NEUROSCALE [Lowe and Tipping, 1996] can be used to produce features for input into a neural network. Its output vectors can be of arbitrarily high dimension. This technique will be reviewed in Chapter 2.

However, for visualization techniques, the dimension of feature space must be two or at most three (given a high performance graphics workstation that can rotate the angle of view at a reasonable speed).

The restriction to two or three dimensions for the output of the visualization techniques opens up possibilities for techniques that would not be computationally tractable for higher dimensions. For example, the *Generative Topographical Mapping* [Svensén, 1998], and its extension to the time

dependent case, discussed in Chapter 4 of this thesis involves forming a distribution based on packing Gaussian distributions into an L dimensional subspace of the input space, arranged on a rectangular lattice in visualization space. The so-called *curse of dimensionality* [Bellman, 1961] precludes using this technique for high values of L, because the computational effort is proportional to N^L where N is the number of lattice points along one of the dimensions.

1.2 Latent variables through time

Techniques for visualization involve:

- Processing high dimensional observation vectors (we define hereafter the dimension of the input vectors to be D), and extracting from them lower dimensional feature vectors (with dimension defined to be L) that encapsulate the relevant information in the data vector.
- Plotting the feature vectors in a suitable form to present the information.

We call the variables of the low-dimensional feature vectors *Latent Variables*, because they may not be observed directly, but only inferred via the observed data vectors. The Latent Variables are supposed to capture the underlying information and structure of the data, while discarding irrelevant information.

We are concerned with the visualization of high dimensional data *through time*, where the data vectors collected are sampled at uniform intervals of time, and are part of a continuous process. Because our prior knowledge of the nature of time-dependent signals indicates that successive data vectors will be correlated, we require that the visualization technique employed should be able to exploit that correlation. It should indicate when an abnormal event has occurred, or alternatively, it should be able to smooth out abnormal data points by taking into account the temporal context.

We shall illustrate informally via a trivial example the kind of benefits we hope to achieve with taking temporal context into account. The example

chosen is best expressed in terms of the Hidden Markov Model formalism explored in Chapter 4, as it involves discretized latent variables.

1.2.1 "Studying the Form" — a simple 1-dimensional example

It is said that of people who bet on horse races, those who "study the form" are supposed to do much better than the average person. The idea behind this is that a particular racehorse will probably go through cycles of being on form and being off form. This is illustrated in Figure 1.2.1, where we have assumed a model where the states of being "off form" and "on form" are of a relatively long duration, whereas the transitions between them are relatively brief.

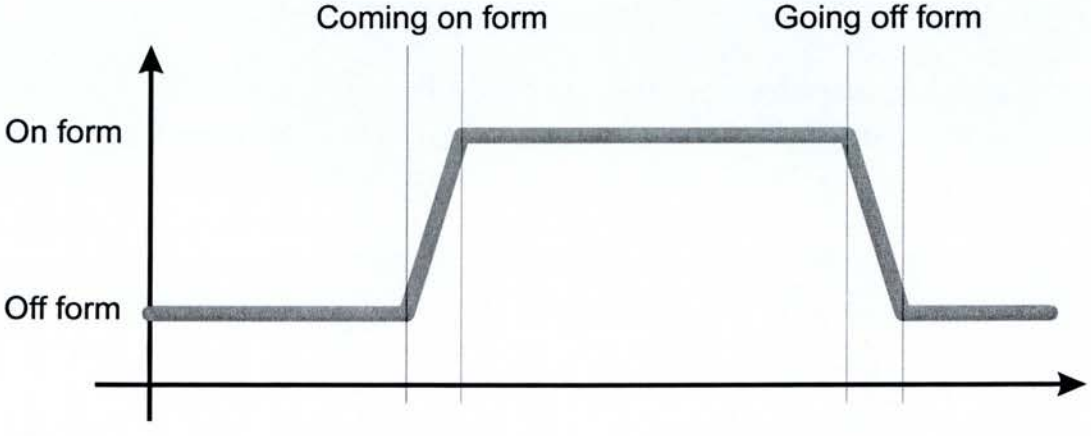


Figure 1.2: Hypothetical graph of the latent variable "form", indicating a possible cycle through which a racehorse might pass. The position within the cycle is supposed to affect the chance that it will win. What we would wish to see in a visualization is this graph, but we only have as observations the sequence of results.

We now choose to model this as a *Markov Chain*, where a system can occupy any one of a number of discrete states, and the next state in the sequence is solely dependent on the previous state. If the form of a racehorse were to be modelled in this way, we could assign four discrete states:

F Off form.

CN **C**oming **on** form.

N **On** form.

GF **G**oing off form.

We assume for the sake of this illustration that the states are sequential (i.e. it cannot revert to a previous state , or skip states), but that the system may stay in one state for more than one time step. This can be represented by a "state transition diagram", as in Figure1.3.

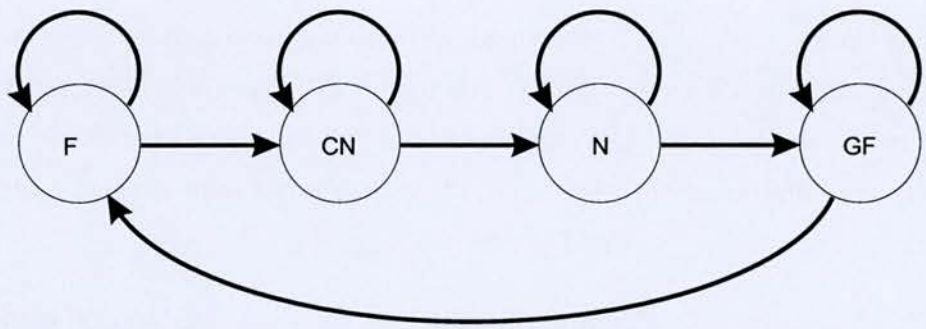


Figure 1.3: State transition diagram for the horse racing example. Four states are (F = Off form), (CN = **C**oming **on** Form), (N = **On** form) (GF = **G**oing off form).

Here each of the circles represents a state of the system, and the arrows represent allowed transitions, between states. An arrow joining a node back to itself represents a time step where the system remains in the same state. Each arrow has an assigned probability, and all the probabilities assigned can be represented in a matrix form, for example as follows:

	F	CN	N	GF
F	0.9	0.1		
CN		0.5	0.5	
N			0.9	0.1
GF	0.5			0.5

where the zeros in the matrix correspond to disallowed transitions, and have been omitted here to emphasize the matrix structure. Such a matrix could then be used to generate a possible sequence of states, from any given starting

state, by taking the row of the matrix corresponding to the current state, and selecting the next state according to the probabilities in that row.

We now further develop the horse-racing example given above. So far the sequence of system states has been modelled as a Markov Chain, via a matrix of transition probabilities. However, the model is not complete for the purposes of making reasonable decisions on whether to make bets or not. The problem is that the "form" of the racehorse in question is a latent variable. We cannot observe or measure the stage in its cycle of being on and off form. All we are able to do is to study the past performance of the animal in recent races. Thus, our system observations are a sequence of results, for example, "Win", "Place"[4], "Lose". From these, we must infer the likely stage the animal is in the cycle, and place bets accordingly. In order to make inferences, therefore we need to know the probability of each outcome, given the current state. These might be assigned as follows:

	Win	*Place*	*Lose*
F	0.1	0.2	0.7
CN	0.4	0.5	0.1
N	0.6	0.3	0.1
GF	0.2	0.4	0.4

Note that the rows of this matrix must sum to unity. To complete the model, we also require a prior probability for each of the states.

Formally, the above probabilistic model is termed a **Hidden Markov Model**, and it is completely specified by three quantities $\{\mathbf{A}, \mathbf{B}, \boldsymbol{\pi}\}$, which correspond to our matrix of state transition probabilities (termed the *transition matrix*), the matrix of outcome probabilities given the state (termed the *emission matrix*), and the prior. The algorithms that will be presented in Chapter 4 formalize a methodology of learning the parameters of the model, and of making inferences of the hidden variables from a sequence of observations. Hence, we note that the phrase "studying the form" is a misnomer; what the person who studies the form is doing in fact is to *infer* the form, from a sequence of observations that are only linked to the hidden variable

[4]In the top three in the race.

via the emission matrix. It is common to represent a Hidden Markov Model diagrammatically as a "trellis" of states as illustrated in Figure 1.4.

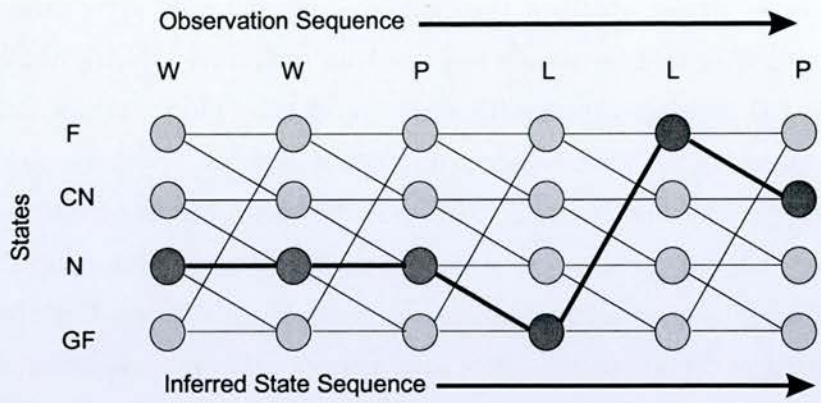


Figure 1.4: Trellis diagram for horse-racing example. Along the top is a sequence of observations (race outcomes). The vertical columns represent the allowed states, and the thin lines between columns represent the allowed transitions. The bold line represents the inferred sequence of states, given the sequence of observations.

1.2.2 What we hope to achieve with a temporal approach.

The latent variable formulation allows this inference of states given observations to be carried out in a principled fashion. We should expect that taking into account temporal context will allow a smoothing out of abnormal data points arising from individual measurements. Consider, for example, the sequence $\{W, W, W, P, W, W, W, L, W, W, W, W\}$. Without temporal context, all we have to make the inference is the emission probability matrix, and therefore, given the isolated loss, we infer that the most likely state for the form is "off form", and for the wins, we infer that it is "on form". However, the isolated loss probably does not mean that the racehorse went off form for one race, and then back on form (in fact according to the transition matrix model that we have used in this model, that sequence of states would be impossible, as the state "off form" can only be reached via the transitional state of "going off form"). Other random factors might account for the loss, such as the quality of the opposition, or if the jockey mishandled the race, and so

forth. The temporal processing allows the most likely sequence of states to be inferred while taking into account the whole observation sequence.

The visualization of the Latent Variable therefore allows one to "get the big picture", and, in this case, infer where we are in the cycle depicted in Figure 1.2.1. It also allows abnormal events to be flagged up, by having a *predictive model* of what the next data point is likely to be. In this case, the predictive model is the transition matrix, which gives, at any time, a prediction of the next likely value of the latent variables, and hence the probability distribution of the next data reading (and hence its expected value). When the new data point arrives, we can compute the probability of that particular event given the observations up to that point, and if it turns out that an unlikely event occurred, we can flag it in some way and take appropriate action (which might be to ignore the event as a freak accident, or it might be to raise an alarm, such as when detecting a sudden and unexpected change in the condition of a patient in a hospital).

1.3 Basic Theoretical Concepts Used

The toy "horse-racing" example discussed in Section 1.2.1 was an example of a **Hidden Markov Model** (HMM), which is one of the tools that will be used in Chapter 4. In this section, we present the broader theoretical framework from which different types of model may be derived, of which the HMM is only one.

1.3.1 Probabilistic Graphical Models

1.3.1.1 Use of graph theory to indicate causal relationships

Probabilistic Graphical Models exploit graph theory to provide a way of representing probabilistic structure. In this thesis we shall consider one particular type of Probabilistic Graphical Model, a **Directed Acyclic Graph**, or DAG, which is also known as a **Bayesian Belief Network**. In such a graph, the nodes represent random variables, and the directed edges indicate a causal relationship between the variables. A probability function $P(x_k|\mathbf{y}_k)$ is associated with each node in the graph, where x_k represents the k^{th} vari-

able, and $\mathbf{y}_k$ represents its parents; i.e. the conditioning variables, which have directed edges ending on x_k. The joint probability distribution of all the variables for such a graph may then be written as the product of the conditional distributions for each node:

$$P(\mathbf{x}) = \prod_{k=1}^{N} P(x_k|\mathbf{y}_k) \tag{1.1}$$

The simplest possible such graph is shown in Figure 1.5.

Figure 1.5: Simplest possible probabilistic graph, indicating a causal relationship between variables x and t.

In the diagram, a causal relationship is indicated between the variable x and the variable t. It expresses the notion that the value of t is *caused*, or *explained* by the value of x. These variables may be discrete, continuous, or vector valued. In terms of the horse-racing example, we might say "The horse lost the race because it was off form". In this case x represents the latent variable "Form" from the previous section, and t represents the observable variable which is the result of a particular race. So we can use the diagram to represent the conditional distribution of results given the form, as $P(t|x)$. However, what we wish to do in practice is to infer the form, given the result; in other words to reverse the direction of the arrow and calculate $P(x|t)$. This may be done by applying Bayes' rule:

$$P(x|t) = \frac{P(t|x)P(x)}{P(t)}. \tag{1.2}$$

1.3.1.2 Conditional Independence and Explaining Away

Consider two separate race results R_1 and R_2, and suppose these were achieved by our racehorse on two successive days. Then we shall expect that the two

variables will not be independent, because we believe that form influences the performance, and that form varies smoothly in cycles. Hence if we know that R_1 is a win, then our expectation that R_2 is also a win or a place is increased. However, suppose we have some inside information that we know will directly affect the performance (for example we are tipped off that the horse has been illegally doped with a performance-enhancing drug). This situation is illustrated by the graph in Figure 1.6.

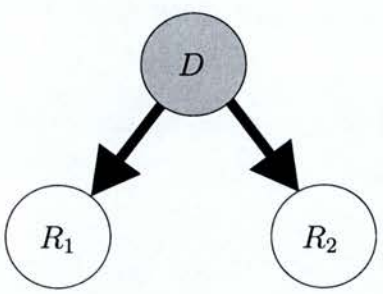


Figure 1.6: R_1 and R_2 are conditionally independent, given the conditioning variable D.

We shade the node marked D to illustrate that the variable (whether or not the horse is doped), is known. In this case, knowledge of R_1 will not affect our expectation of the value of R_2, because the knowledge of the conditioning variable explains the performance; we did not have to infer the conditioning variable from the result set. In this situation, the two variables are said to be *Conditionally Independent.*

Now consider the situation where we have two conditioning variables, each competing to explain the result. Let these be "Form" as before and "Strength of competition". This is represented by the graph in Figure 1.7.

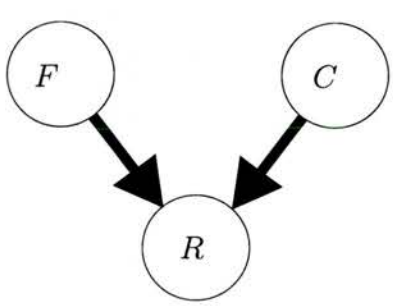
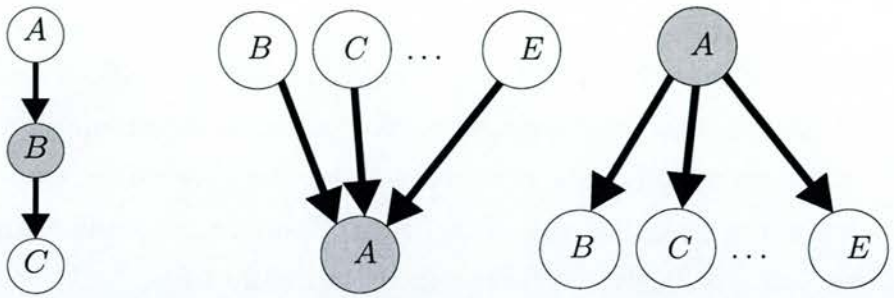


Figure 1.7: Graph representing two possible "causes" for a result.



(a) Serial (b) Convergent (c) Divergent

Figure 1.8: Three types of connection for determining d-separation. If a node is shaded, its value is known.

In this case, in the absence of any knowledge of the race result, we note that F and C are unconditionally independent. Suppose we now know that the horse lost the race. It could either be because it was off form, or that there was an exceptionally strong set of competitors in that particular race. Furthermore, because of the knowledge of R, F and C now become conditionally dependent, conditioned on R. Anything that increases our certainty of F (for example, knowledge that the horse lost the last three races), then decreases the certainty of C. We are now more confident that the horse was off form, and hence we no longer have reason to believe that the competition was strong. This is termed *Explaining Away*.

1.3.1.3 D-separation and Conditional Independence

The phenomenon of conditional independence can be formally represented using directed graphs, in terms of three different types of connection that can be made. These are illustrated in Figure 1.8, where the convention has been adopted of shading in a node whose value is known. In Figure 1.8(a), the arrows indicate a causal connection between A and B and B and C. As long as we do not know anything about the state of variable B, there is an implied causal relationship between A and C. However, if we know for certain the value of B, then there is no implied causal relationship between A and C. In this case, knowledge of B is said to block communication between A and C,

which thereby become independent. They are said to be *d-separated* [Pearl, 1988] . In Figure 1.8(b), if we have no knowledge of the state of A, then the "parent nodes" are independent (as in the above example, where there was no relationship between form and competition). If we know the state of A, then the parents become conditionally dependent. In Figure 1.8(c), the opposite applies. If the state of the parent node is known, then the children become independent.

Two nodes A and B in a network can be said to be d-separated if for all paths between A and B, there exists an intermediate variable V such that either:

- V is either a serial or divergent connection and the state of V is known, or,
- V is a convergent connection and there is no knowledge about the state of either V or any of its "descendent" nodes.

If two nodes of the graph are d-separated, then they are conditionally independent, as it follows that $P(A, B|V) = P(A|V)P(B|V)$. If this is not the case, then A and B are said to be *d-connected.*

The concept of conditional independence is an important one in making probabilistic inferences about a particular variable, given bits of knowledge about all the other variables. In order to write down the joint density of all the variables, we must write a product of conditional probabilities for each node in the graph. In the general case, this will be highly complex, and intractable. One must perform the process of marginalization over all the unknown variables, which will involve the evaluation of multi-dimensional integrals. Because this is an NP-hard problem, approximate methods of inference have to be used. However, in the graphical models in this thesis, all the nodes will have a single parent, and so exact inference will be possible.

If we consider the case of visualization without temporal context, with the vectors of latent variables denoted as $\mathbf{x}_i$ and the observable variables as $\mathbf{y}_i$, then the corresponding probabilistic graphical model for a data set with N variables would be represented as multiple copies of Figure 1.5. Since all the $\mathbf{x}_i$ and all the $\mathbf{y}_i$ are d-separated, there is no problem performing inference

about a particular $\mathbf{x}_i$ given a data measurement $\mathbf{y}_i$, as there is only a single inference in the path. It is conventional to represent multiple independent copies in a Probabilistic Graphical Model using a "plate" notation as in Figure 1.9.

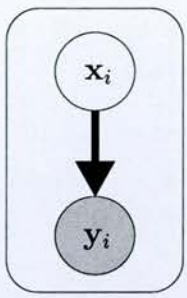


Figure 1.9: Plate representation of multiple independent variables.

In forming a static visualization model we choose a *Prior distribution* $p(\mathbf{x})$ of the latent variables, and a *Conditional distribution* $p(\mathbf{y}|\mathbf{x})$ for the data variables. The parametric form of the conditional distribution includes the fitting parameters that are learned during the training of such a model. In order to get a visualization, we compute the posterior distribution in latent space $p(\mathbf{x}|\mathbf{y})$ using the Bayes' rule, and then plot some convenient statistic derived from that distribution (such as the mean).

In extending to the case where data is time dependent, we shall make the assumption of a "Markov Process", where the state at time t depends only on the state at time $t-1$. Using the same notational convention for variables as in the static case, we can represent this by the graph in Figure 1.10.

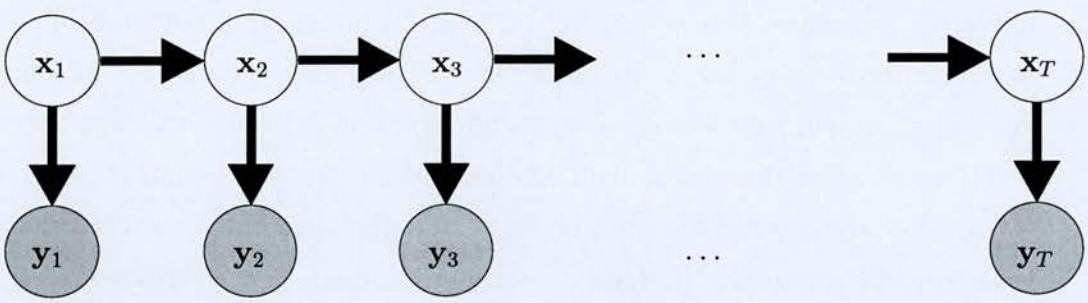


Figure 1.10: Probabilistic Graphical Model representing a Markov Process.

From this graph, we can once again write down straight away the joint distribution of all the variables, given a prior distribution on the first state variable, and a suitable choice for the conditional distribution $p(\mathbf{x}_{n+1}|\mathbf{x}_n)$.

$$p(\{\mathbf{x}\}\{\mathbf{y}\}) = p(\mathbf{x}_1)\prod_{t=1}^{T-1} p(\mathbf{x}_{t+1}|\mathbf{x}_t)\prod_{t=1}^{T} p(\mathbf{y}_t|\mathbf{x}_t). \tag{1.3}$$

In order to perform temporal visualization for data vector $\mathbf{y}_t$, we wish to compute the posterior distribution of the latent variable $\mathbf{x}_t$ given the entire observation sequence, and then plot a suitable statistic from the posterior distribution.

Computation of the posterior distribution involves two steps: a forward chain of inferences from the beginning of the sequence, to calculate the probability of $\mathbf{x}_t$ given the sequence of vectors up to time t, and a reverse chain of inferences, in order to calculate the probability of $\mathbf{x}_t$ given the observation sequence from $t+1$ to T. Given a suitable choice of distributions, this is an analytically tractable problem, which involves $O(N)$ steps.

1.3.2 Learning model parameters using the Expectation–Maximization Algorithm

We discussed in general terms in the last section how to make probabilistic inferences for a visualization model in terms of probabilistic graphical models. We specified the existence of three distributions, which are chosen to construct a particular model, namely the prior distribution $p(\mathbf{x}_1)$ of the latent variables, the conditional distribution $p(\mathbf{x}_{t+1}|\mathbf{x}_t)$ governing the evolution through time of the latent variables, and the conditional distribution $p(\mathbf{y}_t|\mathbf{x}_t)$, which is the mapping from latent (visualization) space to data space. These will be parametric distributions, and their parameters are learnt in a training process, using the *Expectation–Maximization Algorithm* [Dempster et al., 1977]. This is universally referred to as the *EM Algorithm.*

In principle, the fitting of the model parameters is straightforward. In order to get the best model, we maximize the *likelihood* of the model given the data. Likelihood is defined as the probability of the data given the model and the parameters, viewed as a function of those parameters. In practice,

we maximize the logarithm of the likelihood, and taking logs in Equation (1.3), we obtain a sum of terms. We then differentiate with respect to each of the model parameters, equate to zero and solve the resulting equations.

This would be the standard learning approach, if we knew all the data values. But unfortunately, by definition, we do not know the values of the latent variables, and so these have to be treated as "missing data". Accordingly, we have to marginalize over the missing variables, by integrating them out. So, representing the model parameters as θ, we have:

$$\mathcal{L}(\theta) = \log P(\mathbf{y}|\theta) = \log \int_{\mathbf{x}} P(\mathbf{x}, \mathbf{y}|\theta) d\mathbf{x}. \tag{1.4}$$

An elegant justification of the EM algorithm is given in [Roweis and Ghahramani, 1999], based the work of [Neal and Hinton, 1998] along the following lines. We introduce a second, arbitrary distribution Q over the latent variables, and can obtain a lower bound on $\mathcal{L}$:

$$\begin{aligned} \log \int_{\mathbf{x}} P(\mathbf{x}, \mathbf{y}|\theta) d\mathbf{x} &= \log \int_{\mathbf{x}} Q(\mathbf{x}) \frac{P(\mathbf{x}, \mathbf{y}|\theta)}{Q(\mathbf{x})} d\mathbf{x} & (1.5) \\ &\geq \int_{\mathbf{x}} Q(\mathbf{x}) \log \frac{P(\mathbf{x}, \mathbf{y}|\theta)}{Q(\mathbf{x})} d\mathbf{x}. & (1.6) \end{aligned}$$

The inequality obtained by taking the logarithm inside the integral may be proved from Jensen's inequality, and arises from the concave form of the log function. We denote the lower bound as $\mathcal{F}(Q, \theta)$, and rearranging terms we get:

$$\mathcal{F}(Q, \theta) = \int_{\mathbf{x}} Q(\mathbf{x}) \log P(\mathbf{x}, \mathbf{y}|\theta) d\mathbf{x} - \int_{\mathbf{x}} Q(\mathbf{x}) \log Q(\mathbf{x}) d\mathbf{x}. \tag{1.7}$$

The EM algorithm is now the alternation of two steps, the E (for *Expectation*) step, which maximizes $\mathcal{F}$ with respect to the distribution Q, and the M (for *Maximization*) step which maximizes $\mathcal{F}$ with respect to the parameter vector θ:

$$\begin{aligned} Q_{k+1} &\leftarrow \underset{Q}{\operatorname{argmax}}\, \mathcal{F}(Q, \theta_k) \qquad \textbf{E step} & (1.8) \\ \theta_{k+1} &\leftarrow \underset{\theta}{\operatorname{argmax}}\, \mathcal{F}(Q_{k+1}, \theta) \qquad \textbf{M step.} & (1.9) \end{aligned}$$

The maximum of the E step occurs when Q becomes the conditional distribution of $\mathbf{x}$, at which point the bound becomes an equality and the maximum of the M step can be found by maximizing the first term of Equation (1.7), since the second term does not depend on θ. For a proof of this and of Jensen's inequality, see Appendix C.

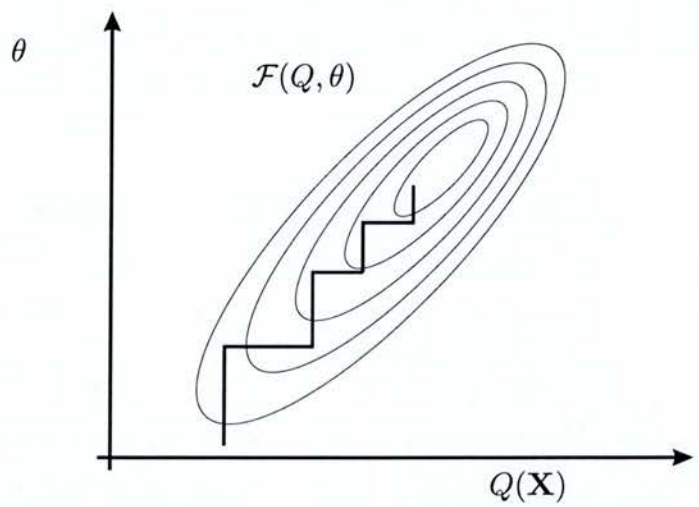


Figure 1.11: EM algorithm as coordinate ascent in $\mathcal{F}$ space. The E steps are the horizontal segments on the path, where $\mathcal{F}$ is maximized with respect to the distribution $Q(\mathbf{x})$, and the M steps are the vertical segments, where $\mathcal{F}$ is maximized with respect to the model parameters θ.

The EM algorithm is then viewed as an alternating coordinate ascent in $\mathcal{F}$, where the E step maximizes the function by varying the distribution $Q(\mathbf{x})$, keeping the model parameters fixed, and the M step maximizes with respect to the model parameters. This is illustrated in Figure 1.11. Since at the beginning of the M step we have that $\mathcal{F} = \mathcal{L}$, the procedure cannot decrease the likelihood.

Finally, we note that we do not necessarily compute the whole distribution in the E step, but just sufficient statistics to describe the distribution as required by the M step. These will be the expected values of the hidden variables and associated variances. Hence the E step, seen in this realization of the algorithm as a maximization in distribution space, is termed the "Expectation" step.

1.4 Outline of the rest of this thesis

The rest of this thesis is arranged as follows:

1. In Chapter 2, we conduct a survey of existing visualization techniques, dividing broadly into two categories, namely those relying on linear mappings (such as Principal Components Analysis), and those using non-linear mappings from data space to visualization space. Potential for the extension of such techniques to the time-dependent case is examined and reviewed.

2. In Chapter 3, we develop a new algorithm for visualization of time dependent data based on the existing technique of Principal Components Analysis. This extends an existing algorithm for the i.i.d. case to the temporal case.

3. In Chapter 4, we develop a new algorithm for visualization of time dependent data using a non-linear mapping of the data space to visualization space. This is based on extending the GTM algorithm of Bishop and Svensen, which itself is a probabilistic development, for the i.i.d. case, of the Kohonen Self-Organizing feature map (SOFM). The extension to the time-dependent case involves treating each such component as a state in a Hidden Markov Model.

4. In Chapter 5, we compare and contrast the performance of the visualization techniques developed on real data, collected from Human Patient Monitoring data, obtained from research at the Oxford University Engineering Department's Signal Processing and Neural Networks Department.

5. In Chapter 6, we summarize what has been developed in the thesis and outline potential future directions of research.

1.5 Nomenclature

Here, we define certain symbols that will be used by convention throughout the thesis.

Term	Dimension	Description
D	Scalar	Dimension of data space
L	Scalar	Dimension of latent (visualization) space
T	Scalar	Number of observation vectors in a sequence
N	Scalar	Number of observation sequences (or number of i.i.d. observation vectors)
$\mathbf{y}$	$D \times 1$	A single observation vector
$\mathbf{x}$	$L \times 1$	A single vector in Latent space
Terms specific to Linear Dynamical Systems		
$\mathbf{A}$	$L \times L$	State dynamics matrix for linear dynamics models
$\mathbf{b}$	$L \times 1$	Linear drift term in state dynamics
$\mathbf{r}$	$L \times 1$	Measurement noise for state dynamics
$\mathbf{S}$	$L \times L$	Measurement noise covariance matrix for state dynamics
$\mathbf{W}$	$D \times L$	Linear mapping between latent and data space
$\boldsymbol{\mu}$	$D \times 1$	Data mean in PCA
$\mathbf{v}$	$D \times 1$	Measurement noise vector in state dynamics
σ^2	Scalar	Variance parameter for isotropic covariance matrix of measurement noise
Terms specific to GTM and GTM through time		
K	Scalar	Number of grid points in GTM prior
M	Scalar	Number of basis functions for GTM mapping
$\mathbf{\Phi}$	$K \times M$	Basis functions evaluated at grid points
$\mathbf{W}$	$M \times D$	Linear map from basis functions to output space
$\mathbf{A}$	$K \times K$	State transition matrix in Hidden Markov Model
$\mathbf{B}$	$K \times 1$	Emission density in Hidden Markov Model
$\boldsymbol{\pi}$	$K \times 1$	Prior distribution in Hidden Markov Model

Common notational conventions are:

Notation	Description
$\mathbf{X}', \mathbf{x}'$	Transpose of a matrix or a vector
$\mathbf{X}^{-1}$	Inverse of a matrix
$\lvert\mathbf{X}\rvert$	Determinant of a matrix
$\lVert\text{expression}\rVert$	Euclidean norm of an expression
$\langle\text{expression}\rangle$	Expectation value of an expression
$\{\mathbf{x}\}_n^m$	Shorthand for the partial observation sequence $\{\mathbf{x}_n, \mathbf{x}_{n+1}, \ldots \mathbf{x}_m\}$
$\{\mathbf{x}\}$	Shorthand for the full observation sequence $\{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_T\}$
$\odot$	Element-wise multiplication of two matrices or vectors

Chapter 2

Survey of Existing Techniques

In this chapter, we perform a survey of existing techniques for data visualization, and, where it has been applied, their extension and applicability to time dependent data.

A visualization of high-dimensional data in $\mathcal{R}^D$ into a lower dimensional space $\mathcal{R}^L$, may be thought of as a mapping, where an L dimensional surface is made to intersect with the D dimensional data space, and the data points are projected by some method onto the L dimensional surface.

Visualizations may be divided into two categories:

- **Linear Mappings**, where the mapping is formed by a linear combination of L axes:

$$\mathbf{x} = \mathbf{W}\mathbf{y}, \qquad \mathbf{W}\epsilon\mathcal{R}^{L\times D} \tag{2.1}$$

- **Non-linear Mappings**, where the mapping is formed by a non-linear function of the data vectors $\mathbf{y}$:

$$\mathbf{x} = f(\mathbf{W}, \mathbf{y}) \tag{2.2}$$

  The mapping may be an explicit function with a closed form (an example being the NEUROSCALE technique of section 2.2.3.2), an inverse mapping (as in the GTM of section 2.2.2), or simply an implicit mapping with no mathematical form, that arises during the learning process (as in the Kohonen SOM of section 2.2.1).

2.1 Linear Mapping Techniques

2.1.1 Principal Components Analysis

Principal Components Analysis (PCA) is the most popular algorithm used for data-visualization based around a linear mapping. Its uses are in fact broader than just for visualization; it can be used for feature selection, representing samples from a high dimensional distribution by a set of feature vectors of lower dimension. The mapping is derived from the discrete form of the **Karhunen-Loéve** expansion [Jollife, 1986], where principal directions (the axes in the visualization) are given by the eigenvectors of the data covariance matrix, and the projections onto these axes are given by the corresponding eigenvalues.

Formally, we expand a random D-dimensional vector $\mathbf{y}$ thus:

$$\mathbf{y} = \sum_{i=1}^{n} x_i \boldsymbol{\phi}_i, \tag{2.3}$$

where the columns of the matrix $\boldsymbol{\Phi} = [\boldsymbol{\phi}_i \ldots \boldsymbol{\phi}_p]$ are taken to be an orthonormal set. The objective is to find a reduced L-dimensional representation:

$$\hat{\mathbf{y}} = \sum_{i=1}^{L} x_i \boldsymbol{\phi}_i, \tag{2.4}$$

that minimizes the mean-squared error between the approximated vectors $\hat{\mathbf{y}}$ and the data vectors $\mathbf{y}$ summed over the data set. It can be shown [Bishop, 1995, pps. 454–456] that the choice that minimizes the error is where the $\boldsymbol{\phi}_i$ correspond to the eigenvectors of the data covariance matrix, corresponding to the largest eigenvalues. This leads to the result that the mean squared error turns out to be the sum of the remaining smallest eigenvalues. The x_i are the associated coordinates in the orthonormal basis defined by the eigenvectors, and are calculated (assuming that the input vectors are zero mean) by computing the dot product of the input vector with the corresponding eigenvector: $x_i = \mathbf{y}^{\mathbf{T}} \boldsymbol{\phi}_i$.

It can also be shown that the PCA projection maximizes the variance of the data along the principal axes. The mapping can be seen as a rotation and scaling of the data after the mean has been subtracted off, as illustrated in Figure 2.1:

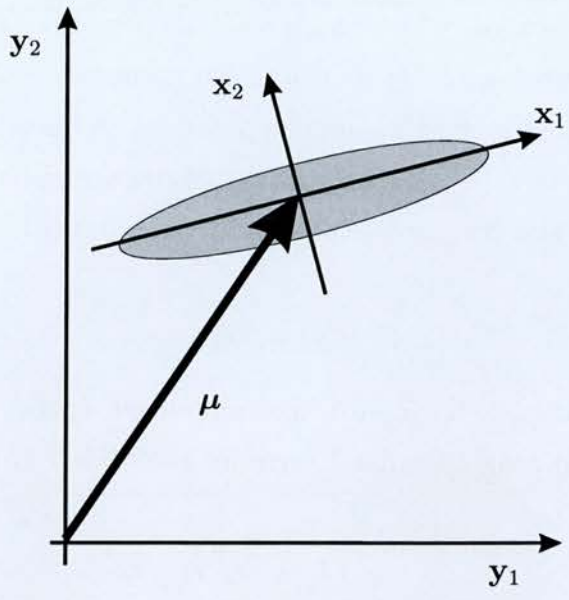


Figure 2.1: Schematic representation of Principal Components Analysis.

2.1.1.1 Computational Requirements of PCA

In section 1.1.2 the relationship between visualization and feature selection was discussed, and it was noted that visualization techniques allowed possibilities for algorithms that are not possible in the general (high-dimensional) feature extraction process, because many algorithms that scale badly with problem dimension can be utilized if the dimension of visualization space is small.

The computational complexity of the standard PCA algorithm requires the computation of the full data covariance matrix which requires $\mathcal{O}(ND^2)$ operations, and the extraction of its eigenvalues and eigenvectors, which requires $\mathcal{O}(D^3)$ operations. Since it is normally the case that $N \gg D$, the first of these terms (the computation of the covariance matrix) is the most expensive in terms of computational requirements.

However, there exist iterative algorithms for computing PCA that only require the computation of the L-dimensional dominant subspace, and these require $\mathcal{O}(NDL)$ operations per iteration, though they require multiple iterations. However for the case $L \ll D$ then it is likely that gains can be made.

In [Oja et al., 1992], a simple iterative algorithm, called the *Subspace*

algorithm is presented, where the linear mapping is represented as a single layer neural network with D inputs and L outputs, and a $L \times D$ weight matrix $\mathbf{W}$. The values of the weights are computed by an iterative algorithm, using a Hebbian update rule. Each iteration i of the algorithm processes a single input vector $\mathbf{x}_i$ and has three stages, namely a forward step:

$$\mathbf{y}_i = \mathbf{W}_i \mathbf{x}_i \tag{2.5}$$

computing a vector in the $L-$dimensional output space; followed by a backward step, computing a feedback term for the update rule:

$$\mathbf{f}_i = \mathbf{W}_i^{\mathbf{T}} \mathbf{y}_i \tag{2.6}$$

and finally the update rule itself:

$$\mathbf{W}_{i+1} = \mathbf{W}_i + \gamma_i \mathbf{y}_i [\mathbf{x}_i - \mathbf{f}_i]^{\mathbf{T}} \tag{2.7}$$

where γ_i is a scalar gain. The algorithm does not have any asymptotically stable vectors, but will tend to converge to an orthonormal basis of the L-dimensional dominant eigenvector subspace of the data covariance matrix. However, this problem can be overcome by introducing an ordered set of weighting parameters for the components of the feedback term $\mathbf{f}$, (giving rise to a weighted subspace algorithm). This algorithm can be adapted to on-line learning, and there are only $\mathcal{O}(DL)$ operations per data vector. However, the convergence properties of the algorithm are not discussed.

In [Tipping and Bishop, 1999], an EM algorithm is given for learning a probabilistic version of PCA (which will be discussed further in Chapter 3 of this thesis). Bishop and Tipping show that the algorithm can be formulated to have $\mathcal{O}(NDL)$ operations per iteration, and that the convergence of the EM algorithm is sufficiently fast that worthwhile gains can be made if $L = 2$ and $D > 20$.

2.1.1.2 Use of PCA for Dimensionality Reduction

A straightforward example of using Principal Components Analysis for dimensionality reduction is given in [Bishop et al., 1993], in which PCA was

used to reduce the dimension of the target vector for a Multi-Layer Perceptron (MLP) neural network [Rumelhart et al., 1986]. The application was to reconstruct the electron density profile across the minor radius of a toroidal vacuum vessel in the JET (Joint European Torus) fusion experiment. Detailed profiles of the electron density, which consisted of 35 points, were only available on a low-frequency basis (1–2 Hz), and the objective was to construct a mapping that would estimate the profile from data that were available on a 10 kHz basis. PCA was used to reduce the dimensionality of the target data from 35 to around 7, which corresponded to the number of degrees of freedom in an existing linear inversion model [Corti, 1980], for the same problem. The neural network was trained to produce the 7-dimensional PCA reduction of the profile from the inputs, and the predicted profiles were then reconstructed using Equation (2.4).

2.1.1.3 Principal Components for Time-Dependent Data

Principal Components Analysis can be used for visualization of time-dependent data much like any other visualization technique. The principal components can be computed in the low (L) dimensional space, and the trajectory can be plotted, instead of a set of points. The advantage of this method is that it involves no change to the computation involved, and it does indeed show time evolution. The obvious limitation is that the computation of the visualization is unable to exploit the fact that successive points are highly correlated in time.

Various attempts can be made to capture the time dependence of the data, by incorporating the time-dependence into the *data preprocessing*:

Windowing. For a scalar time series, $\{x_1, x_2, \ldots, x_n\}$ a sequence of D dimensional tapped delay line vectors can be extracted by applying a rolling window $\mathbf{x}_j = \{x_j, x_{j+1}, \ldots x_{j+D}\}, j = 1 \ldots n - D$. Following this, some form of standard feature extraction is used, usually by frequency based methods, such as FFT or Linear Prediction Coefficients [Makhoul, 1975]. Once the features (typically up to 10 components) have been selected, the resultant sub-space trajectory may be visualized using PCA, or by applying a further

non-linear dimensionality reduction to the PCA components (see Section 2.2.3.2 and [Lowe, 1998]).

Incorporating time as an extra input. This approach, known as **Time-constraint PCA (TC-PCA)** has been adopted in [Reinhard and Niranjan, 1998], where the application was in speech recognition. Given a data set $\mathcal{T}$ consisting of N sequences from time $(1, \ldots, T)$ of D dimensional vectors, represented as $\mathcal{T} = \mathbf{T}_1, \ldots \mathbf{T}_N$ with the sequence $\mathbf{T}_k = \mathbf{y}_{k1}, \ldots, \mathbf{y}_{kT}$, an extra dimension is introduced that represents time explicitly, so $y_{k*}^{D+1} = \tau\,(1, \ldots, T)$. This allows time ordering to be imposed on the visualization. The extent to which time ordering is imposed depends on the parameter τ. Clearly as τ tends to zero, the visualization becomes the same as standard PCA, as the extra time derived input does not contribute significantly to the overall variability of the data. By contrast, if τ is large, then the first principal component will be the time variable itself, and the remaining principal component (assuming $L = 2$) will be what was the first principal component of the initial data. The effect of gradually increasing τ is to "unroll" the data, so the time dependence becomes visible on the graph. This is illustrated in Figure 2.2 for a "toy" data set, where the data vectors are 3-dimensional with the underlying form: $\{\sin(t), 0.5\cos(2t + \frac{\pi}{2}), 0.1\cos(t + \frac{\pi}{2})\}$, to which a small amount of random noise has been added. Sequences of 50 vectors are plotted, so the time input varies from τ to 50τ for each sequence. The plot is given for three different values of τ. As can be seen, the visualization represents a compromise between showing the time dependence, and showing the information from the spatial distribution of the data. In the limit of $\tau \to \infty$, the visualization becomes $L - 1 = 1$ dimensional with respect to the data.

Reinhard and Niranjan adopted a pragmatic approach to the problem of choosing a value for τ. The application is the classification of transitions in speech. The feature vectors were derived from windowed speech data, from which mel-frequency Cepstral coefficients (MFCC) [Davis and Mermelstein, 1980] had been extracted. They defined a difference measure between two trajectories in visualization space, and exhaustively searched for the value of τ that gave the best discrimination. Their results are not as good as classifiers as the standard Hidden Markov Model techniques, but on the other

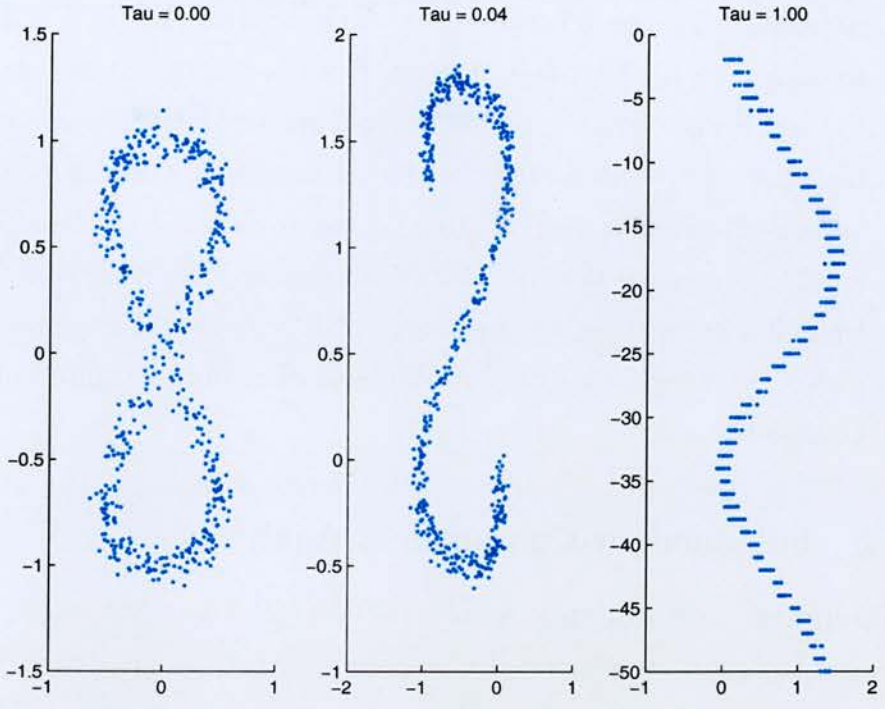


Figure 2.2: Time-constrained PCA **TC-PCA** as proposed in [Reinhard and Niranjan, 1998]. The data consists of three-dimensional data vectors based on the underlying function $\{\sin(t), 0.5\cos(2t + \frac{\pi}{2}), 0.1\cos(t + \frac{\pi}{2})\}$ to which random noise has been added. The left hand plot is equivalent to standard 2-dimensional PCA, with $\tau = 0$. The right hand plot, with large τ means that the time dimension dominates and it is equivalent to a chart-recorder plot of the first principal component. The middle plot is intermediate, showing some time dependence, and some of the contribution from the second principal component.

hand, the model has a greatly reduced number of adjustable parameters, and might therefore be expected to have better generalization properties. The drawbacks of the method are two-fold:

1. The method, as explained, involves throwing away information, by the "sharing" of one of the visualization dimensions with the time variable, resulting in a trade-off for intermediate values of τ, between pure PCA with $L = 2$, which throws away the time information, and a chart plot of the $L = 1$ case.

2. The second drawback is more subtle. PCA is derived by minimizing the

sum-squared reconstruction error. This implies that it is suited best to data with distributions that are described by just two parameters, the mean and variance of the data, namely random variables with a Gaussian distribution. It is clear that the linear ramping of the time variable does not correspond to a Gaussian variable and therefore Principal Components may not be the most appropriate representation. In the following section, the technique of *Independent Components Analysis* is described, which is applicable to data that is significantly non-Gaussian.

2.1.2 Independent Components Analysis

Independent Components Analysis (ICA) [Bell and Sejnowski, 1995] is a related technique to Principal Components Analysis, in that it searches for a linear transformation of the data. However, in contrast to PCA, which only uses second order statistics (derived from the data covariance matrix), ICA is intended to separate out mixtures of signals by use of higher order statistics (principally Kurtosis) of the distributions of the component signals. Essentially, it seeks component signals that are as *statistically independent* as possible. The resultant projection directions are not necessarily orthogonal, as is the case with PCA.

Statistical independence is defined in terms of the ability to factorize the joint probability distribution of all the variables into distributions for each variable independently. Suppose we have a set of random variables $t_1, t_2, \ldots, t_n$, whose joint density is given by $f(t_1, t_2, \ldots, t_n)$. Then we require the following:

$$f(t_1, t_2, \ldots, t_n) = f_1(t_1) f_2(t_2) \ldots f_n(t_n). \tag{2.8}$$

The difference between PCA and ICA may be summarized by the observation that PCA seeks to find the most *faithful* representation of the data, in terms of minimizing the least squares errors in reconstruction, whereas the ICA technique attempts to find *interesting* directions of projection of the data, that will reveal higher order structure. This is illustrated in figure 2.3, where a data set consisting of two elongated Gaussian distributions is

shown. It is clear that the principal axis provides the best representation in terms of least squares error in the reconstruction. However the principal axis projection also hides completely the bi-modal structure of the data.

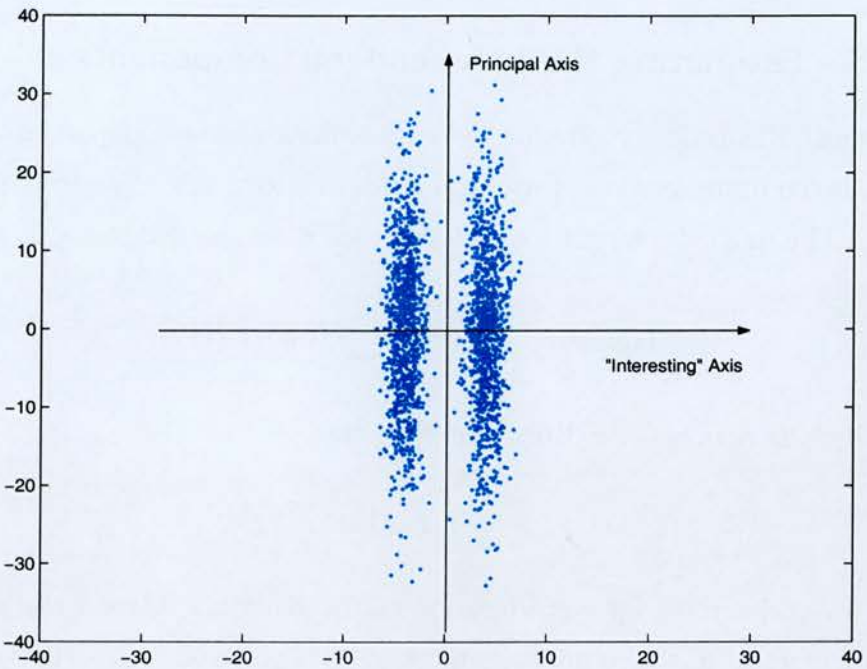


Figure 2.3: Illustrating the conceptual difference between PCA and ICA. The Principal Axis gives the most *faithful* representation, minimizing least squares difference, but hides the clustered nature of the data. The second principal axis, while having lower fidelity, shows more interesting structure in the data.

The degree of "interestingness" in a set of data may be related to the deviation of the underlying probability distribution from a Gaussian; which is considered the least "interesting" (or most "random") distribution. Other distributions exhibit more structure, and it is the aim of ICA to find a linear projection that best reveals that structure. It is also clear that since the product of two Gaussian distributions is itself a Gaussian, that factorization of the probability distribution cannot be performed unambiguously if more than one component is Gaussian[1].

ICA is strongly related to the statistical technique of **Projection Pursuit Analysis** [Friedman and Tukey, 1974], where interesting projection directions are found by optimization of a **projection index**. ICA for noise-

[1]Note, however, that ICA can be performed if only one component is Gaussian.

free data is a special case of Projection Pursuit, where the effective projection index is statistical independence as defined in equation (2.8). A discussion of this is given in [Hyvarinen, 1999, Section 3.4].

2.1.3 Estimating the Independent Components

The most theoretically pleasing way of estimating the independent components is the minimization of **mutual information**, according to [Hyvarinen, 1999]. The mutual information I between m random variables y_i is given by:

$$I(y_1, y_2, \ldots, y_m) = \sum_{i=1}^{m} H(y_i) - H(\mathbf{y}) \tag{2.9}$$

where H denotes the **differential entropy**:

$$H(\mathbf{y}) = -\int f(\mathbf{y}) \log f(\mathbf{y}) \, \mathrm{d}\mathbf{y}. \tag{2.10}$$

This is derivable by consideration of the Kullback–Liebler divergence, a non-symmetric distance measure between two probability distributions:

$$\delta(f_1, f_2) = \int f_1(\mathbf{y}) \log \frac{f_1(\mathbf{y})}{f_2(\mathbf{y})} \, \mathrm{d}\mathbf{y}. \tag{2.11}$$

The statistical independence may be estimated as the Kullback-Liebler divergence between the real density and the factorised density $f_1(y_1)f_2(y_2)\ldots f_m(y_m)$ where the $f_i(.)$ are the marginal densities of the variables y_i. This quantity is identical to the mutual information.

In practice, however, there are difficulties in implementing a direct application of this technique, because the estimation of entropy requires the probability density to be estimated, which is expensive.

The estimation of Independent Components can also be viewed as the maximization of the output entropy of a neural network with outputs that are various non-linear functions $g(.)$:

$$L = H(g_1(\mathbf{w}_1^T\mathbf{x}), \ldots g_m(\mathbf{w}_m^T\mathbf{x})), \tag{2.12}$$

where the $\mathbf{w}_i$ are the weight vectors connected to the output neurons of the neural network.

This leads to the simple stochastic gradient ascent algorithm derived in [Bell and Sejnowski, 1995] for a sigmoidal output unit:

$$\Delta \mathbf{W} \propto \left[\mathbf{W}^T\right]^{-1} - 2\tanh(\mathbf{W}\mathbf{x})\mathbf{x}'. \tag{2.13}$$

The above works well for super-Gaussian distributions[2] , but for sub-Gaussian distributions other functions must be used. Convergence rates for this algorithm are in general very slow, and faster algorithms have been developed more recently, such as **FastICA** ([Hyvirinen and Oja, 1997]).

2.1.4 Applications of ICA

The most important application of ICA is in *blind source separation*, otherwise known as the **cocktail party problem**, where the idea is that a number n of components have been mixed together by a mixing matrix $\mathbf{A}$, to give n signals. The problem is to estimate the unmixing matrix $\mathbf{A} = \mathbf{W}^{-1}$, in order to reproduce the original signals.

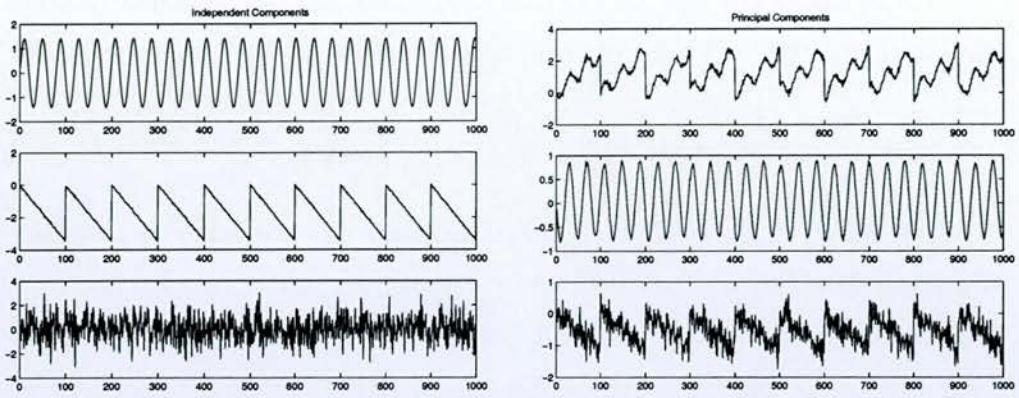
This is illustrated in Figure 2.4, where three original signals; a sinusoid, a sawtooth, and Gaussian noise, were mixed using a 3×3 mixing matrix whose coefficients were generated by sampling a normal distribution with mean zero and unit standard deviation. The mixed signals were processed using the FastICA algorithm, obtained from a public domain source[3].

Figure 2.4(a) shows the three waveforms recovered correctly (though note that the sign is ambiguous in ICA; hence the original sawtooth was of the opposite sense). By contrast, the projections of the Principal components in Figure 2.4(b) do not recover the originals, but shows combinations of all three signals in all three components (though the second principal component is dominated by the sinusoid).

Application of ICA to data visualization is less straightforward than for PCA, because there is no ordering in the independent components as there is in PCA. However, PCA can be used to determine the number of independent

[2]A Super-Gaussian, or *leptokurtic* distribution is one that exhibits positive Kurtosis, and is characterized by having the data more strongly concentrated around the mean than the normal distribution. By contrast, a Sub-Gaussian, or *platykurtic* distribution has negative kurtosis, and correspondingly a flatter peak than the normal distribution.

[3]http://www.cis.hut.fi/projects/ica/fastica/



(a) ICA (b) PCA

Figure 2.4: Comparison of PCA and ICA on a blind source separation problem.

components, [Hyvarinen, 1999, p20], and to reduce the dimension of the input data before application of ICA, in the case of data that is relatively noise-free. This is useful for performing feature selection, and may also be used to perform visualization, if the majority of the variance in the data is concentrated in a few components, so that in order to achieve visualization, selected combinations of the independent components can be plotted.

2.1.5 Temporal dependence in ICA

In [Bell and Sejnowski, 1996], the basic ICA algorithm of Equation (2.13) was applied to the problem of learning the higher-order structure of a natural sound. The sound recorded was a musical tune produced by tapping the teeth, and consisted of a sharp attack of the impact of the tooth tap, (corresponding to a broadband click) followed by a continuous decaying tone. The technique was successfully able to learn the structure of the sound.

However, this paper used the standard ICA algorithm that takes no account of temporal dependence of the data; the time dependence of the audio signal was implicitly represented in the preprocessing, by applying a temporal window to the stream of samples. The data vectors were presented in a random order.

One of the remarkable properties of ICA is that it performs very well without knowledge of the temporal context. Thus, in spite of the fact that

its principal application is in the processing of time-dependent data, it does not attempt to exploit that dependency.

In [Pearlmutter and Parra, 1997], a context-sensitive generalization of ICA was derived. The authors treated the problem of estimating Independent Components in terms of the abstract problem of Density Estimation via Maximum Likelihood. We wish to approximate the true density $p(\mathbf{x})$ by a parametric form $\hat{p}(\mathbf{x};\mathbf{w})$ where $\mathbf{w}$ is an appropriate parameter. The values of the $\mathbf{w}$ elements are estimated by minimizing the Kullback–Liebler divergence:

$$G[p,\hat{p}] = \int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{\hat{p}(\mathbf{x};\mathbf{w})}\, \mathrm{d}\mathbf{x} = H[p] - \int p(\mathbf{x}) \log \hat{p}(\mathbf{x};\mathbf{w})\, \mathrm{d}\mathbf{x} \qquad (2.14)$$

where H is the entropy of the distribution D, and is hence fixed.

An unbiased estimate of G can be obtained by taking a single sample vector:

$$\hat{G} = H[p] - \log \hat{p}(\mathbf{x};\mathbf{w}) \qquad (2.15)$$

From this, a stochastic gradient optimization method can be obtained by computing $d\hat{G}/d\mathbf{w}$ [Robbins and Monro, 1951].

In applying this to the problem of blind-source separation, Pearlmutter and Para consider a particular parametrized form of generative model, where the output vectors $\mathbf{x}$ of the model are the product of a mixing matrix $\mathbf{W}^{-1}$ and an input vector $\mathbf{u}$ of n statistically independent sources, which are drawn from separate parametrized 1-dimensional distributions $f_j(u_j;\mathbf{w}_j)$. The classical ICA algorithm may then be recovered by calculating the derivatives $d\hat{G}/d\mathbf{W}$ and $d\hat{G}/d\mathbf{w}_j$, and applying the stochastic descent procedure.

So far, nothing new has been derived, but the algorithm is then generalized by noting that the distributions $f_j(u_j;\mathbf{w}_j)$ need not be memoryless, but conditioned on the past history of u_j. Hence f_j is of the form:

$$f_j(u_j(t)|\mathbf{u}(t-1),\mathbf{u}(t-2),\ldots;\mathbf{w}_j)$$

Some approximations are now necessary in the general formulation, because changing $\mathbf{W}$ changes the estimate of the recent history of the source term u_j. However, in the paper, the extra terms due to this are dropped; as is common in time series analysis. The authors concluded that the successful results of the analysis implied that the approximation justifiable.

In their experiments, they chose to make the $f_j(.)$ distributions weighted sums of logistic density functions[4]. The scale factors and means of these distributions were allowed to vary, and the means were taken to be linear combinations of the recent history of the u_j. This was therefore a generalization of the original ICA model, which could be retrieved by setting the temporal context to zero, and the number of mixture components to one.

In the experiments, they were able, using the context sensitive ICA, to separate out sources that had strongly Gaussian distributions, which would not have been possible with standard ICA. In the simplest experiment, they took two sources that were drawn from a uniform distribution, and processed them through a low-pass filter. The data had a highly Gaussian histogram, but nonetheless, the algorithm was able to recover the original sources.

In a more involved experiment, they took a set of ten audio signals, and preprocessed them to increase the Gaussianity of the distribution of each source by performing non-linear histogram normalization. The resultant signals could not be separated by the standard ICA algorithm, but the temporal version was able to separate the signals efficiently, even with a very small amount of temporal information.

2.2 Non-linear Mapping Techniques

In the previous section, we dealt exclusively with linear mapping techniques for visualization; in other words the reduced dimension visualization was composed of linear combinations of the variables in data space. If a 2-dimensional visualization is required, this is equivalent to placing a 2-dimensional plane in the D-dimensional data space, and projecting points onto that plane.

With non-linear techniques, we allow a curved 2-dimensional manifold to be used, where the form of the curve is determined either empirically during the training process (as in the Kohonen Self-Organizing Feature Map, or in the Sammon Map), or via a parametric model, whose parameters are determined by the optimization of an objective function.

[4]The logistic density function is derived from the cumulative distribution function $g(t) = 1/(1 + \exp(-t))$, which is derivable from epidemiology.

2.2.1 Kohonen Self-Organizing Feature Map

The Kohonen Self-Organizing Feature Map, [Kohonen, 1989], is one of the most popular non-linear visualization algorithms. It is biologically motivated, creating a mathematical abstraction of the kinds of neural networks found in the brain, in particular the cerebral neo-cortex. Here it has been observed that the neural networks are essentially two dimensional layers of processing units, where each unit (neuron), has many lateral connections to cells in the neighbourhood. Thus excitation of a particular neuron gives rise to a localized area of excitations in surrounding neurons, described as a "bubble of activity". Such patterns of activity have been observed [Kohonen, 1989, pp 126–127] in a raccoon's cortex using a 400 electrode array.

This provides the motivation for the training algorithm of the Self-Organizing Map, in which each of a set of K reference vectors $\mathbf{w}_j$ is associated with a topological position on a grid (usually a 2-dimensional array of nodes, in rectangular or hexagonal spacing[5]). In the initial version of the algorithm, data vectors $\mathbf{y}$ were singly presented to the network, and for each one a "winning node" was computed, being the node for which the Euclidean distance $\|\mathbf{w}_j - \mathbf{y}\|$ was minimized.

Following the calculation of the winning node, the reference vector of the winning node j and all nodes in a neighbourhood around it are updated by moving them proportionally nearer the winning vector, according to the following rule:

$$\Delta \mathbf{w}_i = \eta h_{ij}(\mathbf{w}_i - \mathbf{y}), \tag{2.16}$$

where h_{ij} is the *neighbourhood function*, simulating the lateral interactions between the neurons. This function can take various forms. The simplest is that of a "top hat" which has the value unity up to a certain radius from the winning node, and zero elsewhere. However, smoother mappings can be obtained by having a Gaussian with the maximum at the winning node. The term η is a learning rate.

The training process involves repeated updates of this kind, starting with a relatively large learning rate, and a neighbourhood function whose radius

[5]Empirically, Kohonen and researchers found that a hexagonal array worked best.

extends over the extent of the map. In the early stages of learning, as the neighbourhood is large, the reference vectors are all tightly clustered near the centroid of the data vectors. As learning proceeds, both the neighbourhood size and the learning rate are gradually reduced. As the neighbourhood size reduces, different parts of the map become specialized to different regions of input space. In practice, it has been found with the initial algorithm that it is beneficial to have two stages of training of the network, one which starts with a high neighbourhood size and learning rate, during which both are reduced fast, and a second stage in which the initial neighbourhood size and learning rate are smaller, and they are both reduced more slowly. The second phase of learning amounts to "fine tuning".

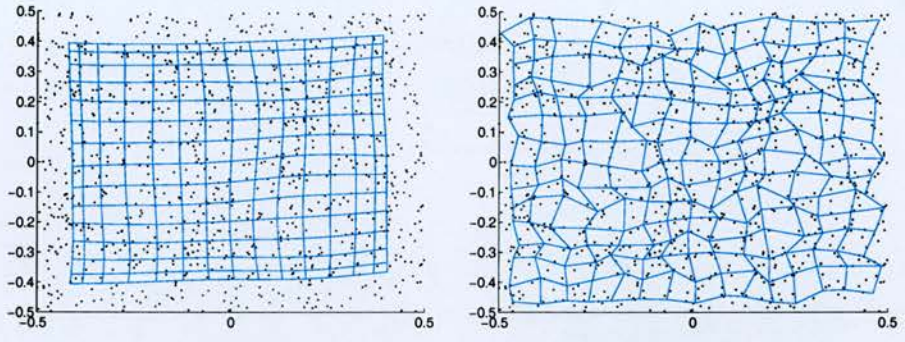
The key property of the Kohonen Map is that it is *topology preserving* in the sense that data vectors that are close together in input space tend to be close together in the final map. In other words, the topological array of reference vectors occupies a smooth 2-dimensional manifold in input space, allowing visualization of the data.

In [Kohonen, 1995], a simpler batch form of the Self-Organizing Map was introduced, which defers the update of the weights until all the data in the training set has been presented. For each input vector $\mathbf{y}_n$, the winning node $j(n)$ is identified, and at the end, the update rule is given by:

$$\mathbf{w}_i = \frac{\sum_n h_{ij(n)} \mathbf{y}_n}{\sum_n h_{ij(n)}}. \tag{2.17}$$

Again, the algorithm is performed iteratively, and the neighbourhood size is reduced in the process of training, but no learning rate is involved.

Figure 2.5 illustrates the topology preserving property of the Self Organizing map. This was a 15×15 map (225 reference vectors) that was trained on 2-dimensional input data vectors $\mathbf{y}_i$ whose values were drawn from a uniform distribution in the range $[-0.5, 0.5]$. Two simulations were performed, with different values for the final neighbourhood size (using a Gaussian function). The results illustrate some of the limitations of the SOFM; it is noted that if the final neighbourhood value is set too large (i.e. too much smoothing) then the map does not extend to the edges of the data (Figure 2.5(a)). This is because of the "surface tension" effect that means that nodes at the edges



(a) Large final neighbourhood (b) Small final neighbourhood

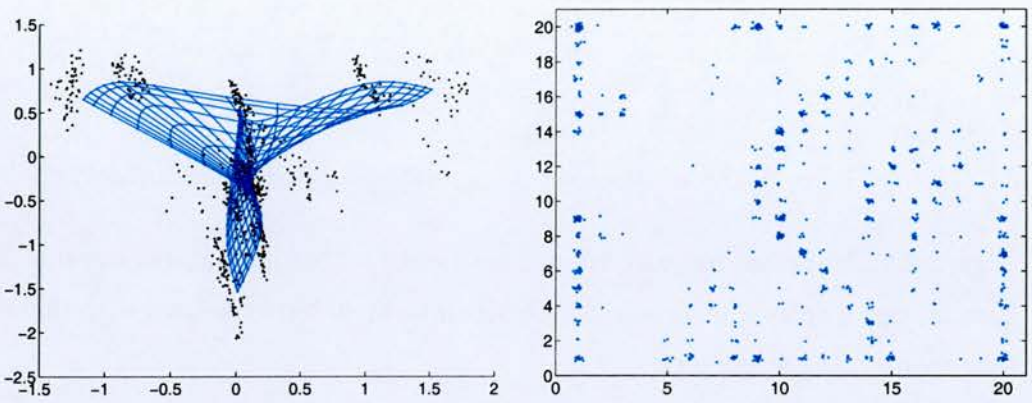
Figure 2.5: Kohonen network trained on random data in the unit square. The two figures display the final map obtained with different settings of the final neighbourhood parameter.

tend to be pulled towards the centre because of the lateral connections with nodes that are closer to the centre. However, Figure 2.5(b) shows that if the final neighbourhood is too small, then each node effectively becomes independent, and the data is overly sensitive to noise in the data. This is typical of the "bias/variance dilemma" [Geman et al., 1992] common in model order selection problems for supervised neural networks; in that if the model is allowed too much freedom, it tends to be too sensitive to noise, and hence generate large changes to small changes in input data. By contrast if it is over-constrained (by having too high a value of the final neighbourhood), it produces a smooth mapping, but with a systematic error (bias).

2.2.1.1 Visualization of data using the Self-Organizing Map

Once trained, the Self-Organizing Map can be used to visualize new data by computing the winning nodes for each data vector and displaying them in a suitable fashion. One drawback of the method is that the map computed is *discrete* and hence only defined at the grid points on the topological map. Hence various heuristics have to be applied if the data set is to be visualized as a whole. One possibility is to plot the data as an intensity histogram over each of the nodes, the height of each node being proportional to the number of times that node was the winner. Another possibility is to represent each

node as a circle, and plot each data point at a random location within its corresponding winning node. Although the randomness does not have any meaning, it allows a quick assessment of the number of data points that were in the cluster associated with the node.



(a) PCA + SOM (b) Flat plot

Figure 2.6: Helicopter data visualized using the Kohonen Self-Organizing map, in PCA projection with the codebook superimposed, and in "flat" visualization space.

Figure 2.6 shows the visualization obtained by the SOM for the helicopter flight data, described in Appendix A, Section A.3. This is flight recorder data obtained from a helicopter on a test flight, with readings such as acceleration, rate of climb, and so forth. Different clusters in the data indicate different modes of flight. In Figure 2.6(a) the data are shown in the standard PCA projection, with the learnt SOM manifold superimposed in the same projection. The clusters at the top of the plot, at the left and right, correspond to when the aircraft was turning. It can be seen that each of the clusters is in fact separated into two, indicating that it performed two different turns, one in each direction, at different turn rates, during the flight. Elongated clusters in the middle along the y axis axis are not well separated, and correspond to climb, level flight, and descent.

Figure 2.6(b) shows the visualization obtained by computing the winning node for each of the data vectors, and then plotting each point at the corre-

sponding topological node position with a random perturbation, to indicate how many data examples were on each node. It appears that the map has adapted to produce better separation of the elongated clusters in the middle of the map, but has lost some of the detail in the outlying clusters corresponding to turns (which appear here at the left hand edge). In the PCA map it is clearly visible that two different turns are present in each direction, but the SOM has lost the distinction and produced two elongated clusters at the end. The PCA projection of the data plus the mesh reveals why this is so; the data is at the edge of the space spanned by training data, and hence this is the region where the map is most stretched out, and is unable to capture the detail. There are also regions where there are very few winners; these correspond to regions of the map that are highly stretched.

Enhancements to the visualization can be achieved by computing the degree of stretching of the map (or "magnification factor"). The computation of the magnification factors is described in [Bishop et al., 1997c], where the technique is originally developed for the GTM algorithm [Bishop et al., 1998b], but is applicable to the batch version of the SOM. Following [Mulier and Cherkassky, 1995], the batch SOM algorithm of equation (2.17) can be reformulated as an iterative kernel smoothing algorithm. This follows from the observation by Mulier and Cherkassky that the value of the neighbourhood function $h(\mathbf{x}_i, \mathbf{x}_j)$ does not depend on the value of the data vector presented, but only on the identity of the winning node. Therefore, partial sums can be made over the vectors assigned to each node j to give an update formula in terms of the mean m_j of the vectors assigned to each node, and a Kernel smoothing function:

$$\mathbf{w}_i = \sum_j K(\mathbf{x}_i, \mathbf{x}_j)\mathbf{m}_j, \tag{2.18}$$

where the Kernel function is given by:

$$K(\mathbf{x}, \mathbf{x}_j) = \frac{N_j h(x, x_j)}{\sum_{j'} N_{j'} h(x, x_{j'})}, \tag{2.19}$$

with N_j being the number of vectors assigned to the j^{th} node. Hence equation (2.18) can be seen to define a continuous and differentiable mapping from latent space to data space. Methods of differential geometry can then be

applied to compute the magnification factors of the map, so that areas of large stretching, indicating segregation between classes, can be highlighted on the map.

2.2.1.2 Extensions to time dependent or sequential data

There are a number of attempts in the literature to generalize the SOM in order to take into account contextual data vectors as part of a sequence.

Adding contextual data to the input vectors A simple technique for incorporating limited contextual information in to the feature vector involves forming compound feature vectors that are a concatenation of the simple feature vectors and their averaged context [Kohonen et al., 1996], [Honkela et al., 1995]. The feature vectors are presented thus:

$$\begin{pmatrix} E(x_{i-1}|x_i) \\ \varepsilon x_i \\ E(x_{i+1}|x_i) \end{pmatrix}. \tag{2.20}$$

This was employed in a two layer SOM tool for browsing large document collections. The first layer was a topological map of related words, where each word in the corpus of words in the documents was assigned a random vector x_i of dimension 90, exploiting the quasi orthogonality of high dimensional vector spaces [Hecht-Nielsen, 1992]. The two dimensional map thus produced showed contextual relationships between words in close proximity on the map, thus demonstrating that the network has learnt the context in sequences of input vectors. The application to browsing large document collections involved using the first map to form a feature vector of a given document, by feeding in the document word by word and constructing a histogram of activations over the document. Such histograms were then used as the feature vectors for a second SOM, which was then able to cluster documents. The system is demonstrated at `http://websom.hut.fi`

A similar approach was used in [Chappelier and Grumbach, 1995], but extending the treatment to extended time series. Here the input is defined to be an evolving sequence of feature vectors through time $\mathbf{x}(t)$, and similarly the weights vectors are treated as functions of time $\mathbf{w}(t)$. In order to define

a time dependent algorithm, they define a distance norm between an input vector sequence and the weights vector as simply the integral of the distance norm over time:

$$\int_t |||\mathbf{x}(t) - \mathbf{w}(t)||| \, \mathrm{d}t \tag{2.21}$$

In the application under consideration (signature verification), the functions over time are discretized as sequences of (x, y) coordinates gathered from a signature tablet, and the distance norm is defined as the standard square of the Euclidean distance: $|||\mathbf{x} - \mathbf{w}||| = \|\mathbf{x} - \mathbf{w}\|^2$. In this case, it reduces to a classical Kohonen network of input dimension $N \times D$ where N is the number of feature vectors in the sequence, and D is the dimension of individual feature vectors. The authors note that if the choice of distance norm were different, then this would result in a non-standard Kohonen Map, but appear to have adopted the standard distance norm in the paper.

Trajectory tracking. This technique involves no modification of the algorithm, but a plotting of the trajectory traced out on the feature map through time. In [Reynolds and Tarassenko, 1993], an application is developed that is intended to assist in learning pronunciation for either foreign language students, or hearing impaired students. The Kohonen network is trained on recorded speech of particular words and phonemes by a normal speaker. Each spoken word was preprocessed into a sequence of feature vectors by windowing and standard frequency analysis techniques generating mel-frequency cepstral coefficients [Davis and Mermelstein, 1980]. Then, in "teaching mode", the pupil is required to pronounce the words, and two trajectories are displayed on the screen; that of the teacher and that of the pupil. This provides a clear visual assessment of the correctness of pronunciation, which is especially useful for deaf pupils who have no normal way of assessing their voices. Reynolds and Tarassenko compute the activation of the j^{th} neuron in the map to the i^{th} feature vector $\mathbf{f}_i$ with the following formula:

$$\mathbf{\Phi}_{ij} = \exp\left(-\frac{\|\mathbf{f}_i - \mathbf{w}_j\|}{v}\right), \tag{2.22}$$

where v is a smoothing parameter, and $\|.\|$ denotes the Euclidean norm. This formulation allows a smoother trajectory to be plotted than just displaying the location of the winning neuron at each point, by calculating all the activations and plotting the centre of mass. It should be noted here that this is a heuristic that is necessary because the Kohonen Map is only defined at discrete points in visualization space. In the more recent GTM technique, [Bishop et al., 1998b], described and extended to the time-dependent case in Chapter 4 of this thesis, a probability distribution is induced in latent space from which meaningful statistics such as the mean can be computed, and displayed as part of the trajectory.

The Temporal Kohonen Map (TKM) This technique [Chappell and Taylor, 1993] is an attempt to build in context sensitivity into the training of the SOM, by a modification of the algorithm, where the activation of each neuron is dependent not only on the current feature vector, but also the previous activations. The activation due to the current feature vector is treated as an electrical potential proportional to the square of the Euclidean distance between the feature vector and the weights vector of the neuron. In addition, a "leaky integrator" term is added, proportional to the previous activation. Hence for the t^{th} input vector, the potential $V_i(t)$ is defined by:

$$V_i(t) = dV_i(t-1) - \frac{1}{2}\|f(t) - w_i\|^2 \tag{2.23}$$

where the constant d is related to the time constant $\tau = 1/(1-d)$.

The paper demonstrated the technique by showing that a TKM was able to learn sequences of 2-d binary vectors, $(00), (01), (10), (11)$. All 16 sequences were presented, and it was shown that, with a value of $d = 0.3$, a 4×4 TKM was able to distinguish between all 16 sequences. However, setting $d = 0$, corresponding to the standard case, led to a network that was only able to distinguish on the basis of the most recent vector presented.

A more complex example was presented in natural language processing, where three sentences were constructed with the same word appearing in different grammatical contexts. A four bit pattern was assigned to each word, and sentences of five words were thus translated into sequences of five binary input vectors. In this case an 8×8 TKM was shown to be able to

find different map positions for the same word in different contexts, even though the word was represented by the same feature vector, whereas the non-temporal map, as expected, had only one place for the same word, as it was not trained by taking temporal context into account.

The Recurrent Self-Organizing Map (RSOM) A slightly different approach is adopted in [Varsta et al., 1997], where leaky integration is applied not simply to the activation of each unit, but to the calculation of the difference vector in the search for the best matching unit. Hence the difference vector between a node and an input is given by:

$$y_i(t) = (1 - \alpha)y_i(t - 1) + \alpha(x_i(t) - w_i(t)) \tag{2.24}$$

where $\mathbf{x}_t$ is the the t^{th} input vector, $\mathbf{y}_t$ is the t^{th} smoothed difference vector, and α is the smoothing constant.

A comparison was made between RSOM and classical SOM for classification of EEG traces in epileptic patients. Wavelet processing techniques were applied to obtain 16-dimensional feature vectors, which were used to train the SOMs. Then each node was labelled according to the classification of the sample (Normal/Epileptic). The classical SOMs were trained using single feature vectors, while the RSOMs were trained using five successive samples, taken from overlapping sample windows from the data. It was noted that improved classification confusion matrices were obtained with the RSOMs indicating that contextual learning had improved the classification performance.

In [Varsta et al., 2000], an analytical comparison was made between the TKM and the RSOM. It was argued that the update rule for the RSOM was more consistent, in using leaky integration for the computation of the difference vectors, and hence temporal context was involved in the update rule, whereas in the TKM it was only used in the computation of the winning unit, and hence the update was biassed towards the last feature vector in the sequence, which was not necessarily the best update direction. The analytic comparison was backed up by simulations to compare RSOM and TKM, and it was found that the TKM tended to distribute most of its weights vectors near the edges of input space, leaving very few for the bulk of the data.

Hidden Markov Model/SOM hybrid systems In [Kurimo, 1997] a hybrid approach is adopted, where a Mixture Density Hidden Markov Model (MDHMM) is used to learn temporal sequences of vectors. The MDHMM is a mixture of Gaussian density functions, with tied covariance matrices over the set. The means of the Gaussian density functions were first initialized by training with the SOM algorithm. The MDHMM was subsequently trained using standard techniques. The application was in phoneme recognition, with a separate MDHMM model trained up for each phoneme. It was found that generalization was improved using the SOM because of the smoothing effect of the topological neighbourhood (as opposed to k-means clustering, which would have single mixtures responding at any given time, rather than a neighbourhood). However, it was also found that better performance could be achieved by applying further discriminative training, using Learning Vector Quantization.

The technique used in this paper has some similarities with the techniques used in Chapter 4 of this thesis, where the probabilistic GTM model is embedded within a Hidden Markov Model framework. However, our approach differs in that the algorithm adapts both the centres and the HMM parameters simultaneously, rather than just using the topographical mapping to initialize the centres.

2.2.2 The Generative Topographical Mapping

The Generative Topographical Map (GTM) [Bishop et al., 1998b] is a probabilistic alternative to the Kohonen Self-Organizing Map. The mathematical details of the algorithm and its extension to the time-dependent case will be covered in Chapter 4 of this thesis.

In the Self-Organizing Map, the code-book of weights vectors arises out of the process of self-organization during the training algorithm. There does not exist an explicit mapping between visualization (latent) space, and the data space, but the self-organization gives rise to the topology preserving property. In the GTM, the mapping from data space to latent space is made explicit by means of a mathematical function, giving rise to a smooth mapping. The code-book vectors are then the results of applying that mapping to a regular

lattice of points in latent space. These points in data space are taken as the centres of a Gaussian Mixtures Density model, which consists of spherical Gaussians, which in the standard model are constrained to have the same variance and mixing coefficients. The model is "trained" by maximizing the likelihood of the data, using an EM algorithm. A suitable form of the mapping function is chosen (an RBF network) so that the steps of the EM algorithm are tractable. The mapping from latent to data space is illustrated schematically in Figure 2.7.

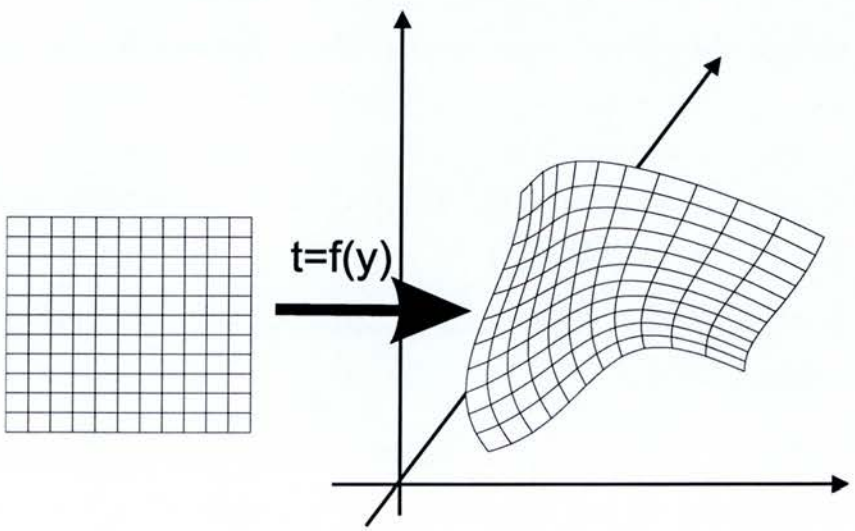


Figure 2.7: Schematic representation of the GTM mapping. A smooth function is defined to map between latent space (on the left) to data space (here represented as 3-D). The curved manifold fits through the data in data space, and lattice points on it are the centres for a Gaussian mixtures model, where each mixture has the same variance and mixing coefficient.

Visualization of data using the GTM is related by considering the posterior probability of each data point in latent space (evaluated via Bayes' theorem). Since the full probability density function would give too much information if it were computed for each data point, it is normal to compute some useful statistic (such as the mean) for each data point.

Differences between the SOM and the GTM are summarized as follows:

- GTM is a probability density function, whereas SOM is not.

- In GTM, the topology preserving property is specified in the functional form at the start, whereas in SOM it arises naturally in the heuristic training algorithm.
- In the GTM, the user-specified parameters directly control the "stiffness" of the mapping, whereas in SOM, the user parameters are not directly related to the final form of the map, and thus more empirical experimentation would be necessary to obtain the desired results.
- In SOM, there is no explicit function optimization taking place. Although the discretization error tends to decrease during training, it does not necessarily decrease monotonically. By contrast GTM uses an explicit function (log likelihood), which is maximized by the EM algorithm [Dempster et al., 1977], which is guaranteed to increase the likelihood at each step.
- SOM produces a discretized visualization, whereas GTM produces a continuous one.

A recent paper [Haese and Goodhill, 2001], proposes a technique for reducing the number of user specified parameters in the SOM training. One of the problems with the SOM is the specification of the initial learning rate, neighbourhood size, and their rate of decay. Haese and Goodhill propose a Kalman Filter implementation of the SOM, where the state vectors are the weights vectors, which are updated by the Kalman filter state-transition matrix (which is the unit matrix in this case) at each learning step. The predicted new state is then computed from the input (the randomly selected training example), and a predicted output (the activation) is computed. The Kalman filter gain matrix is then used in place of the learning coefficient in order to provide the weight update. This provides an automatically estimated learning rate for each weight vector. Additionally, a recursive parameter estimation procedure is employed (also using a Kalman filter) to update the width of the neighbourhood function at each step. A comparison was performed with the GTM on a standard data set used in [Bishop et al., 1998b] to show that Auto-SOM produces similar results to GTM in terms of quantization error, and topology preservation. It was concluded that Auto-SOM

was a competing alternative to GTM, in allowing the elimination of user-selected coefficients. However, it should be borne in mind that Auto-SOM at the end is no different from a standard SOM; the only distinction being in the training. Thus it does not provide a density estimate, or a continuous visualization of the data. The kind of visualization achieved is the same as given in Figure 2.6. In the comparison between GTM and SOM in the paper, the results for the two algorithms are presented as histograms of the number of hits on winning neurons.

In Chapter 3 of this thesis, the Kalman filter is used to extend the PCA algorithm to the time dependent case. It should be noted that this is a different approach to the above, in that the Kalman filter is used to track the evolution of the *data vectors* through time, whereas in Auto-SOM, the Kalman filter is used as a model for the evolution of the learning parameters and weight vectors through time. The data vectors are still presented to the network in a random order.

2.2.2.1 Extensions of GTM to time dependent case

This is dealt with extensively in Chapter 4 of this thesis. To date, the only other example of the use of GTM in a time dependent sense is in [Reinhard and Niranjan, 1998], where a similar pre-processing to the Time Constraint PCA of Section 2.1.1.3 was applied to the speech recognition problem. The pros and cons of this approach have been discussed under that section, and are similar for its application to the GTM. The algorithm used is essentially still a static training algorithm, while one of the components of the feature vector is time.

2.2.3 Sammon Map and Neuroscale

2.2.3.1 Sammon Map as a Topology-Preserving Mapping

The Sammon Map [Sammon, 1969], is a popular non-linear topology preserving mapping. It is performed by minimizing an error function, termed STRESS which attempts to preserve inter-point distances in the visualization space:

$$E = \frac{1}{\sum_i \sum_{j<i} d^*_{ij}} \sum_i \sum_{j<i} \frac{[d^*_{ij} - d_{ij}]^2}{d^*_{ij}}, \tag{2.25}$$

where d^*_{ij} is the Euclidean distance between points in input space, and d_{ij} is the Euclidean distance between points in visualization space.

The minimization of this function involves computing the derivatives of the STRESS function with respect to each of the points in visualization space. A suitable gradient descent technique should be employed. Drawbacks of the algorithm are:

- The algorithm works on the raw data, and computes explicit mappings for each data point. There is no parametric representation of the mapping produced, and hence it cannot compute generalizations for new data.
- The computational effort and storage space required scales as N^2 where N is the number of data points. This places severe constraints on the size of data set that can be analyzed.

In practice, a pragmatic approach is adopted in overcoming this complexity problem, in pre-clustering the data using a classical algorithm such as K-means, and then performing the Sammon Map on the prototype vectors thus obtained. This suggestion was originally made by Sammon, who had noted that with the computers of the time there was a practical limit of around 200 data points. With modern computers this can be increased to a few thousand[6]. However, the technique of pre-clustering means that individual data points cannot be visualized, as the map is just a lookup table between input point and mapped point.

2.2.3.2 NEUROSCALE as a functional form of Sammon Map

In [Tipping, 1996], a new architecture, NEUROSCALE was proposed, that overcomes the first of these problems, and also allows the above clustering approach to be performed on the data. This architecture expresses the topology preserving mapping in terms of a parametric function $\mathbf{y} = (\mathbf{x}, \mathbf{W})$, where

[6] A problem with 1000 data points stored in double precision would require 8 MB of storage, and 4000 data points would require 128MBytes.

$\mathbf{W}$ is a parameter vector that is tuned by an optimization algorithm to minimize the value of the stress function. The mapping chosen is a Radial Basis Function network [Broomhead and Lowe, 1988], with the centres and widths set up by standard methods (such as K-means clustering), and the output layer weights trained by a novel training algorithm **Shadow Targets** [Tipping and Lowe, 1998]. The key idea behind this algorithm is to provide, at each iteration, a set of "estimated" targets for the RBF network, based on "shadowing" the standard Sammon gradient descent approach. Hence given current positions of the map $\mathbf{y}_i$, the new targets at each step are given by:

$$\hat{\mathbf{y}}_i = \mathbf{y}_i - \eta \frac{\partial E}{\partial \mathbf{y}_i}, \tag{2.26}$$

where η is a learning rate parameter that is required because in the early stages of training, when stress has a high value, the full step length can lead to an increase in the value of stress.

The NEUROSCALE architecture has an advantage over the Sammon Map in that it provides an explicit functional mapping from data space down to visualization space[7]. As a result, it is possible to use the model to obtain visualizations for new data. This overcomes to a certain extent the scaling problems inherent in the Sammon approach; it is not necessary to use all the data to train the model. Furthermore, pre-clustering of the training data could be performed for very large data sets, and then the mapping used to perform visualizations of individual data points.

The normal way one would reduce the number of training points for NEUROSCALE would be to pre-cluster the data using K-means, but this is not the only possible technique. In Fig. 2.8, we show how NEUROSCALE may be used to assist in the visualization of the results of a SOM mapping. The flattening out produced by a SOM map can lead to enhancement of details, but can also hide structure, by smearing out sharp features, because many code-book vectors are attracted to these regions. If we use NEUROSCALE and train it on the code-book vectors, then the resultant map will place those vectors that are close in data space correspondingly close in the 2-D

[7]Note that this is in contrast to the GTM which provides a functional mapping from visualization space to data space, and the visualization is then obtained by calculating the posterior distribution. In NEUROSCALE there is no probabilistic model.

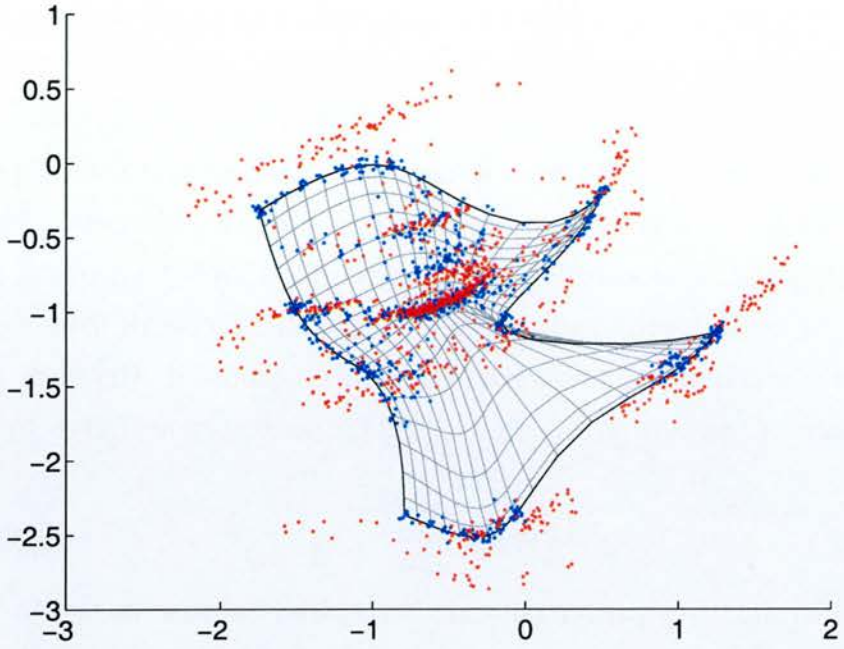


Figure 2.8: SOM codebook vectors and data, for the helicopter data, visualized using NEUROSCALE

visualization. It can also help to show where the map has folded over. The blue points correspond to the winning nodes in the map, with a random scatter to indicate the number of points (as in the flat plot in Fig. 2.6(b)). The red points are the actual data points, projected through the same NEUROSCALE map. It would, of course be possible to derive a similar map just using the data to train the NEUROSCALE map, which in this case would be feasible from the point of view of storage, as the data set consists of just 914 points. However, in this case, the alternative "flat" visualization would not be available. This way of visualization using NEUROSCALE in conjunction with the SOM gives a useful alternative method of looking at the results. Particularly, in this plot, it becomes clear where the SOM is not fitting the data well (at the edges of the plot), thereby explaining why the SOM loses some of the detail. It should also be noted from a practical point of view, that the training of the NEUROSCALE map took a fraction of the time of the training of the SOM, although if all the data, rather than the code-book vectors were used to train NEUROSCALE, that the computation time would be much longer, because of the quadratic scaling law of NEUROSCALE.

There are no current extensions of NEUROSCALE to incorporate time-dependent data, though it has been used to analyze time series data by incorporating the time-dependence into the pre-processing of the data. In [Lowe, 1998] the technique was used to give visualization of EEG data from a vigilance monitoring experiment. The pre-processing consisted of constructing tapped delay vectors, and then embedding these vectors using both PCA and ICA as dimension reducing techniques. The embedded delay vectors were then visualized via NEUROSCALE.

2.3 Summary

In this chapter, we have reviewed techniques for data visualization in general, and considered the extensions of such techniques to time-dependent data. The techniques reviewed fall into the categories of Linear and Non-Linear projection methods.

The most common approaches to dealing with time dependent data appear to be:

1. Incorporating time explicitly as an extra input, as in the Time-constraint PCA, [Reinhard and Niranjan, 1998]. This technique has the merit of being easy to implement and not requiring any changes to existing algorithms. Its disadvantage is that it uses up one of the latent variables for the time input, and hence throws away information.

2. Independent Components Analysis is an alternative linear technique to PCA. Its underlying principle is to find axes of projection (not necessarily orthonormal) that enable the joint probability distribution of the components to be factorised. It thereby finds "interesting" directions of projection, that capture structure. It has been effectively applied to processing time-dependent data, both in its standard form, which does not exploit temporal correlation, and in a time dependent form [Pearlmutter and Parra, 1997], via a probabilistic formulation taking into account the last few data vectors. While the technique is highly effective in time-dependent processing (such as in blind source separation), it has some limitations as a visualization tool, in that there

is no ordering of the Independent Components (whereas there is with Principal Components). Hence if there are more than L independent components, visualizations can only be achieved by plotting combinations of them.

3. Incorporating context by concatenating feature vectors from adjacent time-slices. This has primarily been applied to standard Kohonen maps, ([Kohonen et al., 1996], [Chappelier and Grumbach, 1995]. This is a simple approach that works effectively, but has the disadvantage that it only allows a limited amount of temporal context to be inserted, because for each contextual point, we increase the dimension of the code-book vectors by the size of input space.

4. Incorporating context by suitable preprocessing of tapped delay line vectors, [Lowe, 1998]. This is also a means of preprocessing of the data vectors, and was applied to the NEUROSCALE technique in the cited paper. This technique is applicable when, for example, there is a single input, but we wish to capture the temporal dependence. Again, it requires the length of the feature vector to be proportional to the number of samples in each sequence.

5. Use of a hybrid architecture where a non-time-dependent algorithm is used to initialize a standard time dependent feature such as the Hidden Markov Model In [Kurimo, 1997], the standard Kohonen map was used to initialize the HMM.

6. Incorporating local context effects in the training, via heuristic methods based on leaky integrators [Chappell and Taylor, 1993], [Varsta et al., 1997]. Again, these were adaptations of the standard Kohonen Map.

A common feature with the above approaches is that the length of time sequence that can be accommodated is somewhat limited, either because only a limited past context is processed (as in context sensitive ICA, or in the leaky integrator techniques applied to Kohonen maps), or because it would lead to prohibitively large feature vectors (by making a composite feature vector consisting of a sequence of individual vectors).

In contrast to the above approaches, the following two chapters derive algorithms that are intended to process time sequences of vectors of arbitrary length (in principle), where full account is taken of the time sequence in a probabilistic model. We shall also exploit at each time step the future context as well as the past. This involves a model where the feature vector at time t is conditioned both on the previous examples $1 \ldots t-1$ and the future examples $t+1 \ldots T$.

Chapter 3

Probabilistic PCA Through Time

3.1 Introduction

In this chapter, we introduce a latent variable model for visualization through time, based on Principal Components Analysis (PCA), which was reviewed in Chapter 2. This builds upon the existing technique of Probabilistic PCA (PPCA) [Tipping and Bishop, 1997], which is a probabilistic extension of standard PCA for the case of i.i.d. data. In that model, the prior distribution of the latent variables, which are the projections of the D-dimensional data into the L-dimensional principal subspace, is an L-dimensional spherically symmetric multivariate Gaussian, with zero mean and unit variance.

In our model, the prior is replaced by the conditional distribution $p(\mathbf{x}_{n+1}|\mathbf{x}_n)$, whose mean evolves according to linear dynamics, and which thereby attempts to track the temporal sequence of data. Because temporal context is taken into account, the predictive model for the next latent space point in a sequence should form a much more localized distribution than for the static case, which will allow a greater amount of smoothing out of non-time related artefacts, and better detection of anomalous data than for the i.i.d. case. This model will turn out to be a special case of the linear Kalman Filter.

3.2 Probabilistic PCA for the static case

In this section, we review the basic theory of PPCA for the case of i.i.d. data. This will then be extended to the time dependent case.

Standard Principal Components analysis is a linear projection of D-dimensional vectors $\{\mathbf{y}_n\}, n \in \{1, \ldots N\}$ onto the *principal axes* $\mathbf{w}_j, j \in \{1, \ldots L\}$. The principal axes are given by the so-called "dominant eigenvectors" of the data covariance matrix $E[(\mathbf{y} - \boldsymbol{\mu})(\mathbf{y} - \boldsymbol{\mu})']$, i.e. the eigenvectors associated with the largest eigenvalues. Visualization is performed by projecting the mean-adjusted data vectors onto the principal axes:

$$\mathbf{x}_n = \mathbf{W}'(\mathbf{y}_n - \boldsymbol{\mu}) \tag{3.1}$$

PCA not only maximizes the variance in the lower dimensional space, but also minimizes the reconstruction error, which is the sum of squared distances between the projected points in the principal subspace, and the original data vectors $\sum_n \|\mathbf{y}_n - \mathbf{W}\mathbf{x}_n - \boldsymbol{\mu}\|^2$. A proof of this property is given in [Bishop, 1995, pp 454–456].

However, standard PCA is limited in that it is not a probabilistic model, and is thus poor at detecting anomalous data. While it can show when a data point varies in an out-of-plane direction, it will not produce any indication of novelty if the data is in plane, but well removed from the body of data.

Probabilistic PCA addresses this problem by incorporating the transformation matrix $\mathbf{W}$ within a density model, that is closely related to the latent variables method known as *Factor Analysis*. In this model, the mapping is given by the equation:

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \boldsymbol{\mu} + \boldsymbol{\epsilon}, \tag{3.2}$$

where $\mathbf{W}$ is a $D \times L$ matrix of *factor loadings*, and $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{\Psi})$, a noise model for the latent variables. In classical Factor Analysis $\boldsymbol{\Psi}$ is chosen to be a diagonal matrix. The motivation behind this approach is so that the components of the data are conditionally independent given the latent variables. There is no analytic solution in the general case for the factor analysis latent variable model, which therefore has to be solved by an iterative EM algorithm.

In probabilistic PCA, a further constraint is placed on the noise covariance matrix so that it is a multiple of the unit matrix, hence $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2\mathbf{I})$. Although PPCA is thus a special case of Factor Analysis, the motivations behind the two methods are different. In PPCA, the variances in the data vectors are supposed to be common to all of them, whereas in Factor Analysis, there is some common variance, and some individual variance. The Latent Variables are supposed to give individual explanatory causes for the data, to which a score can be assigned, for individual data components. PCA, by contrast, is simply a projection that does the best possible job in explaining all the variability of the data, and therefore is suited to visualization.

Given the isotropic Gaussian prior, the probability distribution over data space for a given value of the latent variable $\mathbf{x}$ is given by:

$$p(\mathbf{y}|\mathbf{x}) = (2\pi\sigma^2)^{-D/2} \exp\left\{-\frac{1}{2\sigma^2}\|\mathbf{y} - \mathbf{W}\mathbf{x} - \boldsymbol{\mu}\|^2\right\} \tag{3.3}$$

The prior distribution of the latent variables is chosen to be a fixed Gaussian:

$$p(\mathbf{x}) = (2\pi)^{-L/2} \exp\left\{-\frac{1}{2}\mathbf{x}'\mathbf{x}\right\} \tag{3.4}$$

Because both the prior and the conditional distribution are Gaussians, this allows analytic evaluation of the integral to marginalize over the latent variables, to give the distribution in $\mathbf{y}$:

$$\begin{aligned} p(\mathbf{y}) &= \int_{\mathbf{x}} p(\mathbf{y}|\mathbf{x})p(\mathbf{x})d\mathbf{x} \\ &= (2\pi)^{-D/2}|\mathbf{C}|^{-1/2} \exp\left\{-\frac{1}{2}(\mathbf{y} - \boldsymbol{\mu})'\mathbf{C}^{-1}(\mathbf{y} - \boldsymbol{\mu})\right\}, \end{aligned} \tag{3.5}$$

where the covariance matrix $\mathbf{C} = \sigma^2\mathbf{I} + \mathbf{W}\mathbf{W}'$. Using Bayes' rule, the *posterior distribution* in Latent Space may be constructed, also as a Normal distribution,

$$\begin{aligned} P(\mathbf{x}|\mathbf{y}) &= (2\pi)^{-L/2}|\sigma^2\mathbf{M}|^{1/2} \times \\ &\exp\left[-\frac{1}{2}\left\{\mathbf{x} - \mathbf{M}^{-1}\mathbf{W}'(\mathbf{y} - \boldsymbol{\mu})\right\}'(\sigma^{-2}\mathbf{M})\left\{\mathbf{x} - \mathbf{M}^{-1}\mathbf{W}'(\mathbf{y} - \boldsymbol{\mu})\right\}\right] \end{aligned} \tag{3.6}$$

with mean $\mathbf{M}^{-1}\mathbf{W}'(\mathbf{y}-\boldsymbol{\mu})$ and Covariance matrix $\sigma^2\mathbf{M}^{-1}$, where $\mathbf{M} = \sigma^2\mathbf{I} + \mathbf{W}'\mathbf{W}$.

In [Tipping and Bishop, 1997] it is shown that the maximum likelihood estimate for the parameters of the density model of Equation (3.5) may be obtained when the columns of $\mathbf{W}$ span the principal subspace of the data, and where the variance is given by:

$$\sigma^2 = \frac{1}{D-L}\sum_{i=L+1}^{D}\lambda_i, \tag{3.7}$$

where the λ_i are the $D-L$ smallest eigenvalues of the data covariance matrix, as used in standard PCA. This gives a simple intuitive interpretation of the density model as a "Gaussian Pancake", whose plane is given by the matrix $\mathbf{W}$ in the principal subspace of the data, and where the σ^2 parameter accounts for the remaining out-of-plane variance in the model.

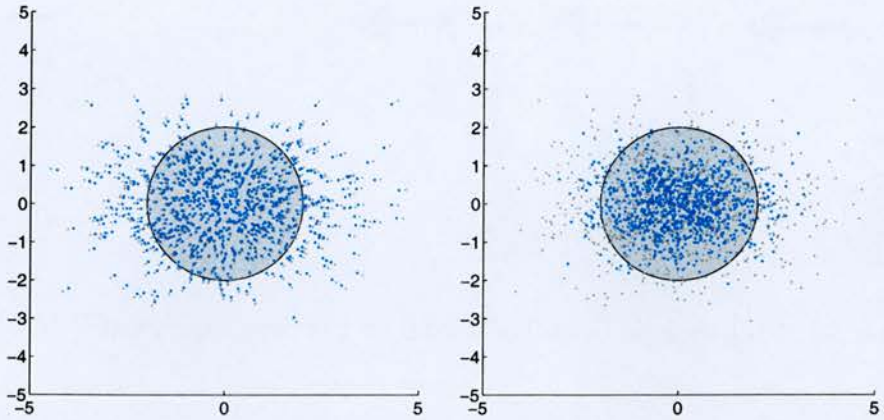
Visualization of the data is achieved by plotting the means of the posterior distribution of Equation (3.6):

$$\langle\mathbf{x}\rangle = (\sigma^2\mathbf{I} + \mathbf{W}'\mathbf{W})^{-1}\mathbf{W}^{\mathbf{T}}(\mathbf{y}-\boldsymbol{\mu}) \tag{3.8}$$

The effect of the σ^2 term Equation (3.8) is to shrink the data towards the origin with respect to standard PCA. The larger the value of σ^2 the greater the degree of shrinkage. The effect of shrinkage is illustrated in Figure 3.1, where two different 3-dimensional Gaussian pancake distributions have been plotted, one where the variance along the least significant axis is small, and one where it is considerably larger, giving rise to a larger value of σ^2. The larger value gives rise to a greater degree of shrinkage (Posterior means are plotted in blue, and the standard PCA projection is plotted in grey). The grey disk represents the 2σ boundary of the prior.

The greater degree of shrinkage for the model with the larger value of σ^2 has a nice intuitive explanation; the poorer the explanatory capabilities of the model, the greater the reliance on the prior. In Figure 3.1(b), the value of σ^2 is greater, i.e. there is a larger amount of variance unexplained by the linear model. Hence the prior dominates.

It should also be noted that there is no necessity for the columns of $\mathbf{W}$ to be orthogonal, only that they should span the principal subspace. In



(a) Small σ (b) Large σ

Figure 3.1: PPCA shrinks distribution towards the origin. Probabilistic PCA representations of two Gaussian pancake distributions, generated from a 3-dimensional spherical Gaussian, with the axes scaled by (1,1.5,0.2) in (a) and by (1,1.5,0.7) in (b). The standard PCA projection is shown in grey, and the PPCA projection is in blue, with the 2 standard deviation contour of the Prior shown as a disk. The right hand diagram, which gives rise to a larger value of σ^2 in the density model, has a larger shrinkage towards the origin.

[Tipping and Bishop, 1997], an iterative EM algorithm is given for PPCA which is not guaranteed to give orthogonal vectors, though an orthogonal basis set can be retrieved by standard techniques. If the columns are made to be orthogonal, then it can be seen that Equation (3.8) reduces to the standard PCA projection as σ^2 tends to zero.

3.3 Probabilistic PCA through time

We now formulate an algorithm that extends the probabilistic PCA algorithm to the time-dependent case, which will be referred to as **Probabilistic PCA Through Time** or **PPCATT**. The time-dependent case may be represented by the Probabilistic Graphical model shown in Figure 3.2,where the shaded nodes are the observed data vectors $\mathbf{y}_t$, and the unshaded nodes are the latent variables $\mathbf{x}_t$.

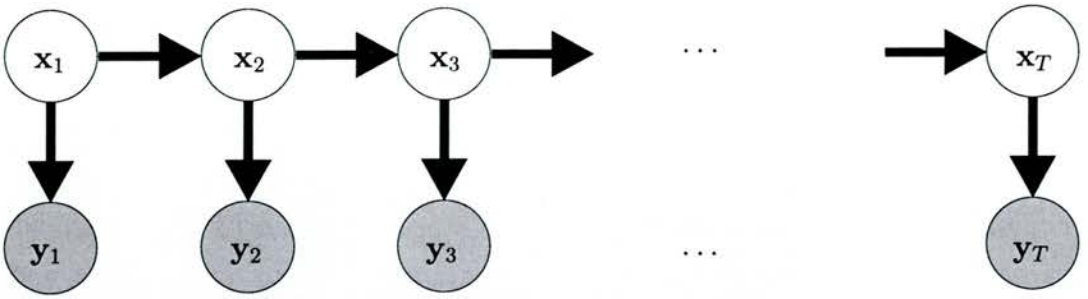


Figure 3.2: Probabilistic Graphical Model for the time dependent PCA algorithm.

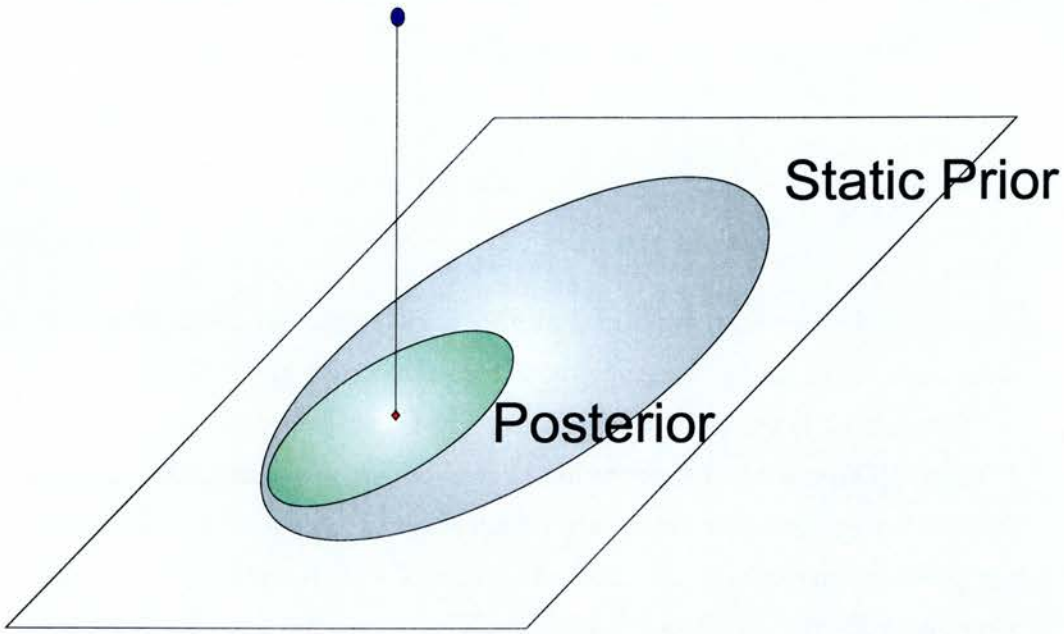
In PPCATT, the static prior distribution of Equation (3.4) is replaced at each time step by a conditional distribution $p(\mathbf{x}_{t+1}|\mathbf{x}_t)$. For the first time step, a static prior is used. At each step, the posterior distribution $p(\mathbf{x}_t|\mathbf{y}_t)$ is used to calculate the prior for the next stage given this conditional distribution. This constitutes a time-update step, which predicts the next value of the latent variables, and hence for the next value of the observable data $\langle \mathbf{y} \rangle_{t+1}$. The difference between this prediction and the observed data vector $\mathbf{y}_{t+1}$ is then used to compute the posterior distribution for $\mathbf{x}_{t+1}$. The difference between the static and dynamic models is thus illustrated diagrammatically in Figure 3.3.

The motivation behind this is that the posterior distribution of the latent variables, and hence the new prior for the next state will have a much lower variance than in the static case, allowing detection of anomalous data, and also a smoothing to take place via the response time of the linear dynamical system. This is because the more localized prior distribution, based on the previous time step will bias the posterior distribution towards itself.

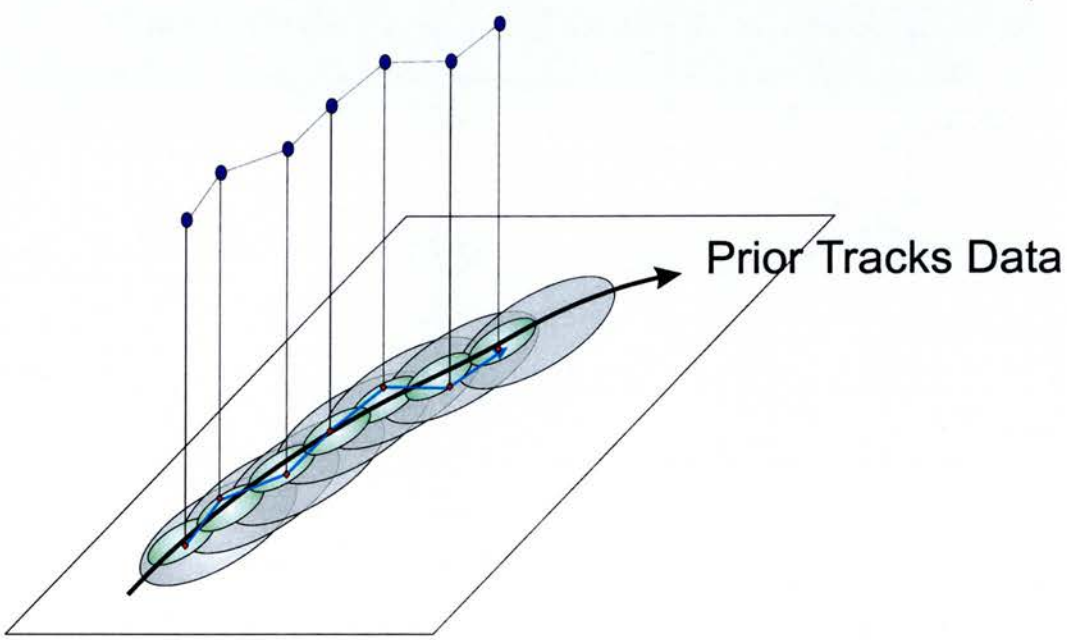
Using the conditional independence properties of the variables implied by the absence of edges to the graph of Figure 3.2, we may write out the joint distribution of all the variables $\mathbf{x}$ and $\mathbf{y}$ as follows:

$$p(\{\mathbf{x}\}_1^T, \{\mathbf{y}\}_1^T) = p(\mathbf{x}_1) \prod_{t=1}^{T-1} p(\mathbf{x}_{t+1} \,|\, \mathbf{x}_t) \prod_{t=1}^{T} p(\mathbf{y}_t \,|\, \mathbf{x}_t). \tag{3.9}$$

The basic model that we choose in PPCATT involves the assumption that the state $\mathbf{x}$ evolves according to a linear dynamical system, with additive



(a) PPCA



(b) PPCATT

Figure 3.3: Comparison of Probabilistic PCA for the i.i.d. case and for the time dependent case. In (a), the i.i.d. case is illustrated, where a single static prior is used for the latent distribution. In the dynamic case (b), the static prior is replaced by a conditional distribution $p(\mathbf{x}_{t+1}|\mathbf{x}_t)$ which evolves through time as a linear dynamical system.

Gaussian noise. The linear nature of PCA implies also that the output is a linear combination of the state variables with additive noise as well:

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{b} + \mathbf{r} \tag{3.10}$$

$$\mathbf{y}_t = \mathbf{W}\mathbf{x}_t + \boldsymbol{\mu} + \mathbf{v}, \tag{3.11}$$

where $\mathbf{r}$ and $\mathbf{v}$ are state and measurement noise vectors, distributed normally with zero mean and covariance matrices given by $\mathbf{S}$ and $\sigma^2\mathbf{I}$.

Our model is essentially very similar to that derived in [Ghahramani and Hinton, 1996a], with the exception that the measurement noise vector is distributed as an isotropic Gaussian, rather than one with a diagonal covariance, because we are deriving a temporal version of PPCA, rather than a temporal version of Factor Analysis. Additionally, we wish to compute explicitly the constant terms $\mathbf{b}$ and $\boldsymbol{\mu}$; the former so as to be able to model a system that had constant drift (for example in a condition monitoring application), and the latter so as to be able to model data with a non-zero mean.

We specify the three probability distributions to use in the model as follows:

$$p(\mathbf{y}_t|\mathbf{x}_t) \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2\mathbf{I}) \tag{3.12}$$

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t) \sim \mathcal{N}(\mathbf{A}\mathbf{x}_t + \mathbf{b}, \mathbf{S}) \tag{3.13}$$

$$p(\mathbf{x}_1) \sim \mathcal{N}(\boldsymbol{\pi}_1, \mathbf{V}_1). \tag{3.14}$$

Thus the model is fully specified by the parameter set:

$$\boldsymbol{\theta} = \left\{\mathbf{W}, \boldsymbol{\mu}, \sigma^2, \mathbf{A}, \mathbf{b}, \mathbf{S}, \boldsymbol{\pi}_1 \mathbf{V}_1\right\}. \tag{3.15}$$

The algorithm for learning the model parameters will be an EM algorithm. The basic philosophy of the EM algorithm has already been discussed in Chapter 1. The algorithm is formulated by writing down the log likelihood in terms of all the variables, replacing the hidden values by their expected values, given the current parameter set. These values are computed by the E step of the algorithm, which involves using the Kalman Smoothing algorithm, which for each time step computes the posterior distribution of the latent

variables, given the entire observation sequence. In the M step, sufficient statistics from this distribution are substituted into the expression for log likelihood, and standard differential calculus methods are used to perform the maximization. The process is repeated until convergence.

3.3.1 Derivation of the M step

In order to compute the M step equations, we write down the expression for the Expected Complete Data Log Likelihood (i.e. the expected value of the log likelihood if we knew the values of the latent variables):

$$\mathcal{L} = \left\langle \ln p(\mathbf{x}_1) + \sum_{t=1}^{T-1} \ln p(\mathbf{x}_{t+1} \mid \mathbf{x}_t) + \sum_{t=1}^{T} \ln p(\mathbf{y}_t \mid \mathbf{x}_t) \right\rangle . \tag{3.16}$$

We then differentiate with respect to each of the model parameters, equate to zero, and solve. As each of the terms is a sum of squares, this is a straightforward solution expressible in terms of linear regression.

The derivation that follows is very similar to the outline proof given in [Ghahramani and Hinton, 1996a], with the differences that the update formula for the isotropic output noise covariance parameter σ^2 is new. Also we explicitly include the constant terms $\mathbf{b}$ and $\boldsymbol{\mu}$, and discuss their impact on the implementation of the equations.

However, the presence of the constant terms $\boldsymbol{\mu}$ and $\mathbf{b}$ leads to a messy derivation with large numbers of terms in the equations, and so for clarity, we follow the recommendation in [Roweis and Ghahramani, 1999], of introducing an $(L+1)^{\text{th}}$ state variable whose value is always unity, which allows the $\mathbf{b}$ term to be absorbed into an augmented state dynamics matrix thus:

$$\mathbf{A}_+ = \begin{pmatrix} \mathbf{A} & \mathbf{b} \\ 0 & 1 \end{pmatrix}, \tag{3.17}$$

and the data mean $\boldsymbol{\mu}$ to be absorbed as an extra column in the output mapping matrix:

$$\mathbf{W}_+ = \begin{pmatrix} \mathbf{W} & \boldsymbol{\mu} \end{pmatrix} \tag{3.18}$$

Hence now the linear system reduces to the standard model given in [Roweis and Ghahramani, 1999], with the augmented quantities substituted:

$$\mathbf{x}_{t+1}^{+} = \mathbf{A}_{+}\mathbf{x}_{t}^{+} + \mathbf{r} \tag{3.19}$$

$$\mathbf{y}_{t} = \mathbf{W}_{+}\mathbf{x}_{t}^{+} + \mathbf{v}. \tag{3.20}$$

We should, however, note at this point that introducing the extra constant state variable results in a Singular Normal Distribution for $p(\mathbf{x}_{t+1}^{+}|\mathbf{x}_{t}^{+})$, because the corresponding covariance matrix only has non-zeros in $L \times L$ top left sub-block, and zeros elsewhere. In practice, this is not a problem, as it can be regarded as a limiting case where the variance of the extra state variable tends to zero. Thus the distribution may be factorized as a product of the original distribution and a delta function distribution:

$$p(\mathbf{x}_{t+1}^{+}|\mathbf{x}_{t}^{+}) = (2\pi)^{-L/2}|S|^{-1/2} \exp\left\{-\frac{1}{2}(\mathbf{x}_{t+1} - \mathbf{A}\mathbf{x}_{t} - \mathbf{b})'\mathbf{S}^{-1}(\mathbf{x}_{t+1} - \mathbf{A}\mathbf{x}_{n} - \mathbf{b})\right\} \boldsymbol{\delta}(x_{t}^{(L+1)} - 1), \tag{3.21}$$

which will give mathematically the same result when expectations are calculated, because integration over a delta function simply picks out the corresponding value.

The simplified equations allow a more succinct derivation to be given. We first derive the update equations for the data space parameters, which only involve the third term in (3.16), which (omitting the constant term) expands as:

$$\mathcal{L}_{\text{Data}} = \frac{DT}{2}\ln\sigma^2 - \sum_{t=1}^{T}\frac{1}{2\sigma^2}\left\langle(\mathbf{y}_{t} - \mathbf{W}_{+}\mathbf{x}_{t}^{+})'(\mathbf{y}_{t} - \mathbf{W}_{+}\mathbf{x}_{t}^{+})\right\rangle. \tag{3.22}$$

Differentiating with respect to $\mathbf{W}_{+}$:

$$\frac{\partial\mathcal{L}_{\text{Data}}}{\partial\mathbf{W}_{+}} = \sum_{t=1}^{T}\frac{1}{2\sigma^2}(\mathbf{y}_{t} - \mathbf{W}_{+}\langle\mathbf{x}_{t}^{+}\rangle)'\langle\mathbf{x}_{t}^{+}\rangle = 0 \tag{3.23}$$

hence the solution for $\mathbf{W}_{+}$ is:

$$\mathbf{W}_+ = \left(\sum_{t=1}^{T} \mathbf{y}_t \langle \mathbf{x}_t^{+\prime} \rangle\right) \left(\sum_{t=1}^{T} \langle \mathbf{x}_t^{+\prime} \mathbf{x}_t^{+} \rangle\right)^{-1}. \tag{3.24}$$

This is the standard Moore–Penrose pseudo-inverse solution for multiple linear regression, with a constant term in the right hand column of the data matrix.

For the output noise, we differentiate with respect to σ^2:

$$\frac{\partial \mathcal{L}_{\text{Data}}}{\partial \sigma^2} = -\frac{DT}{2\sigma^2} + \frac{1}{2\sigma^4} \sum_{t=1}^{T} \langle (\mathbf{y}_t - \mathbf{W}_+ \mathbf{x}_t^+)' (\mathbf{y}_t - \mathbf{W}_+ \mathbf{x}_t^+) \rangle = 0 \tag{3.25}$$

whence:

$$\begin{aligned} \sigma^2 &= \frac{1}{DT} \sum_{t=1}^{T} \left(\mathbf{y}_t' \mathbf{y}_t - 2\mathbf{y}' \mathbf{W}_+ \langle \mathbf{x}_t^+ \rangle + (\mathbf{W}_+ \langle \mathbf{x}_t^+ \rangle)' \mathbf{W}_+ \langle \mathbf{x}_t^+ \rangle \right) && (3.26) \\ &= \frac{1}{DT} \sum_{t=1}^{T} \left(\mathbf{y}_t' \mathbf{y}_t - \mathbf{y}_t' \mathbf{W}_+ \langle \mathbf{x}_t^+ \rangle \right). && (3.27) \end{aligned}$$

We now derive the re-estimation equations for the latent space parameters $\{\mathbf{A}_+, \mathbf{S}_+\}$ governing the evolution through time of the latent space distribution. First, we expand the distribution for $\mathbf{x}_{t+1}$ given $\mathbf{x}_t$:

$$p(\mathbf{x}_{t+1}^+ \,|\, \mathbf{x}_t^+) = (2\pi)^{-L/2} |\mathbf{S}|^{-1/2} \times \exp\left\{ -\frac{1}{2} (\mathbf{x}_{t+1}^+ - \mathbf{A}_+ \mathbf{x}_t^+)' \mathbf{S}_+^{-1} (\mathbf{x}_{t+1}^+ - \mathbf{A}_+ \mathbf{x}_t^+) \right\}. \tag{3.28}$$

The term $\mathbf{S}_+$ denotes the augmented covariance matrix with the desired covariance matrix $\mathbf{S}$ in the top left $L \times L$ sub-block, and zeros elsewhere, except for $\mathbf{S}_+(L+1, L+1) = \epsilon^2$, which will yield a singular normal distribution in the limit as $\epsilon^2 \to 0$.

As before, we proceed by taking logs and the expectation of the resultant expression. We then differentiate with respect to each of the variables to be re-estimated to obtain the update equations. The expression to be maximized is:

$$\mathcal{L}_{\text{State}} = -\frac{1}{2} \ln |\mathbf{S}| + \left\langle \sum_{t=1}^{T-1} -\frac{1}{2} (\mathbf{x}_{t+1}^+ - \mathbf{A}_+ \mathbf{x}_t^+)' \mathbf{S}_+^{-1} (\mathbf{x}_{t+1}^+ - \mathbf{A}_+ \mathbf{x}_t^+) \right\rangle \tag{3.29}$$

Differentiating this with respect to $\mathbf{A}_+$ gives:

$$\frac{\partial \mathcal{L}_{\text{State}}}{\partial \mathbf{A}_+} = \left\langle \sum_{t=1}^{T-1} (\mathbf{x}_{t+1}^+ - \mathbf{A}_+ \mathbf{x}_t^+)' \mathbf{S}_+^{-1} \mathbf{x}_t^+ \right\rangle = 0. \tag{3.30}$$

Noting that we can find an exact equivalent solution by diagonalizing $\mathbf{S}_+$, and rotating the coordinates into a frame where the covariance is the unit matrix, and subsequently transforming back, we can drop the $\mathbf{S}_+^{-1}$ term, and, rearranging terms get the solution:

$$\mathbf{A}_+ = \left(\sum_{t=1}^{T-1} \left\langle \mathbf{x}_{t+1}^+ \mathbf{x}_t^{+\prime} \right\rangle \right) \left(\sum_{t=1}^{T-1} \left\langle \mathbf{x}_t^+ \mathbf{x}_t^{+\prime} \right\rangle \right)^{-1}, \tag{3.31}$$

which is exactly analogous to the output matrix solution (3.24), in that the solution is the standard linear regression fit using the expected values of the latent variables, and is therefore in the form of a pseudo-inverse.

For the state noise covariance, we differentiate (3.29) with respect to $\mathbf{S}_+^{-1}$ to get:

$$\begin{aligned} 0 = \frac{(T-1)}{2} \mathbf{S}_+ - \\ \frac{1}{2} \sum_{t=1}^{T-1} \left(\left\langle \mathbf{x}_{t+1}^+ \mathbf{x}_{t+1}^{+\prime} \right\rangle - \mathbf{A}_+ \left\langle \mathbf{x}_t^+ \mathbf{x}_{t+1}^{+\prime} \right\rangle - \left\langle \mathbf{x}_{t+1}^+ \mathbf{x}_t^{+\prime} \right\rangle \mathbf{A}_+' + \right. \\ \left. \mathbf{A}_+ \left\langle \mathbf{x}_t^+ \mathbf{x}_t^{+\prime} \right\rangle \mathbf{A}_+' \right). \end{aligned} \tag{3.32}$$

The last two terms cancel because from (3.10) $\mathbf{A}_+ \left\langle \mathbf{x}_t^+ \right\rangle = \left\langle \mathbf{x}_{t+1}^+ \right\rangle$ and the noise term $\mathbf{r}$ vanishes when the expectation is calculated.

Hence the solution for the state noise covariance matrix, after rearrangement of terms is given by:

$$\mathbf{S}_+ = \frac{1}{T-1} \left(\sum_{t=1}^{T-1} \left\langle \mathbf{x}_t^+ \mathbf{x}_t^+ \right\rangle - \mathbf{A}_+ \sum_{t=1}^{T-1} \left\langle \mathbf{x}_t^+ \mathbf{x}_{t+1}^+ \right\rangle \right). \tag{3.33}$$

Finally, we need the re-estimation equations for the initial state vector and covariance matrix. Expanding the log likelihood term for the prior distribution, we have the expression:

$$\mathcal{L}_{\text{Prior}} = -\frac{1}{2} \left\langle (\mathbf{x}_1^+ - \boldsymbol{\pi}_1^+)' \mathbf{V}_{1+}^{-1} (\mathbf{x}_1^+ - \boldsymbol{\pi}_1^+) \right\rangle - \frac{1}{2} \ln |\mathbf{V}_{1+}| \tag{3.34}$$

whence:

$$\frac{\partial \mathcal{L}_{\text{Prior}}}{\partial \boldsymbol{\pi}_1^+} = \left(\langle \mathbf{x}_1^+ \rangle - \boldsymbol{\pi}_1^+\right) \mathbf{V}_{1+} = 0 \tag{3.35}$$

giving:

$$\boldsymbol{\pi}_1 = \langle \mathbf{x}_1^+ \rangle \tag{3.36}$$

For the initial state covariance, we have:

$$\frac{\partial \mathcal{L}_{\text{Prior}}}{\partial \mathbf{V}_{1+}} = \frac{1}{2}\mathbf{V}_{1+} - \frac{1}{2}\left(\langle \mathbf{x}_1^+ \mathbf{x}_1^{+\prime} \rangle - \langle \mathbf{x}_1^+ \rangle \boldsymbol{\pi}_1^+ - \boldsymbol{\pi}_1^+ \langle \mathbf{x}_1^+ \rangle + \boldsymbol{\pi}_1^+ \boldsymbol{\pi}_1^{+\prime}\right) = 0 \tag{3.37}$$

and substituting in for $\boldsymbol{\pi}_1^+$, yields:

$$\mathbf{V}_{1+} = \langle \mathbf{x}_1^+ \mathbf{x}_1^{+\prime} \rangle - \langle \mathbf{x}_1^+ \rangle \langle \mathbf{x}_1^{+\prime} \rangle \tag{3.38}$$

This last equation requires a small modification for the case of multiple observation sequences (which is otherwise straightforwardly obtainable by averaging the expected quantities obtained in the E step over the N observation sequences). However for the initial state covariance, the straightforward substitution of $\boldsymbol{\pi}_1^+$ will no longer simplify in the same way. Hence:

$$\mathbf{V}_{1+} = \langle \mathbf{x}_1^+ \mathbf{x}_1^{+\prime} \rangle + \sum_{i=1}^{N} \left(\langle \bar{\mathbf{x}}_1^+ \rangle \langle \bar{\mathbf{x}}_1^{+\prime} \rangle - \langle \mathbf{x}_1^+ \mathbf{x}_1^{+\prime} \rangle \langle \bar{\mathbf{x}}_1^+ \rangle - \langle \bar{\mathbf{x}}_1^+ \rangle \langle \mathbf{x}_{1,(i)}^+ \rangle' \right) \tag{3.39}$$

Then, by completing the square in the bracketed term, we obtain the solution:

$$\mathbf{V}_{1+} = \langle \mathbf{x}_1^+ \mathbf{x}_1^{+\prime} \rangle - \langle \mathbf{x}_1^+ \mathbf{x}_1^{+\prime} \rangle \langle \mathbf{x}_1^+ \mathbf{x}_1^{+\prime} \rangle' + \frac{1}{N} \sum_{i=1}^{N} \left[\langle \mathbf{x}_{1,(i)}^+ \rangle - \langle \bar{\mathbf{x}}_1^+ \rangle \right]' \left[\langle \mathbf{x}_{1,(i)}^+ \rangle - \langle \bar{\mathbf{x}}_1^+ \rangle \right] \tag{3.40}$$

3.3.1.1 E-step for PPCATT

The solution to the E-step involves finding solutions to the so-called *filtering/smoothing* problem. First we give the equations for the straightforward case where there is no drift term $\mathbf{b}$ from the linear dynamics equation (3.10).

Following [Ghahramani and Hinton, 1996a], we introduce the symbols $\mathbf{x}_t^t$ to denote $E(\mathbf{x}_t|\{\mathbf{y}\}_1^t)$ and $\mathbf{V}_t^t$ to denote $\mathrm{Var}(\mathbf{x}_t|\{\mathbf{y}\}_1^t)$.

In the filtering problem, we compute the distribution of the latent variables at time t, $p(\mathbf{x}_t \mid \{\mathbf{y}\}_1^t)$, i.e. the posterior probability of the latent variables given the observation sequence up to and including time t. This will involve forward recursion through the data, which are the standard Kalman Filter equations.

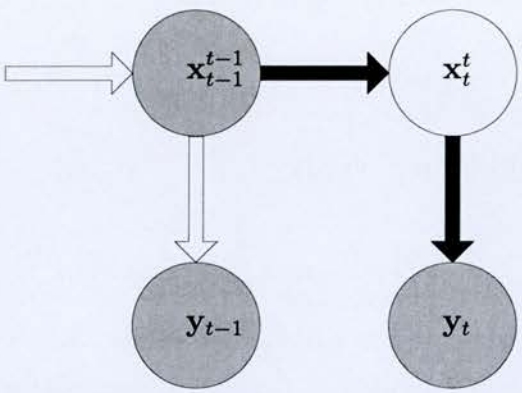


Figure 3.4: Fragment from Probabilistic Graphical Model illustrating the forward (Kalman Filter) recursion

This is illustrated as a fragment of a probabilistic graphical model in Figure 3.4. The key to the recursion is that the current *a posteriori* estimate of the distribution of $\mathbf{x}$ breaks the chain in the graph, so that the next step in the inference chain only depends on $\mathbf{x}_{t-1}$ and the next measurement (via the d-separation criteria introduced in Chapter 1).

The forward recursions break down into a *time update step*, for the mean and error covariance of $\mathbf{x}$, followed by a measurement update.

The time update equations are as follows:

$$\mathbf{x}_t^{t-1} = \mathbf{A}\mathbf{x}_{t-1}^{t-1} \tag{3.41}$$

$$\mathbf{V}_t^{t-1} = \mathbf{A}\mathbf{V}_{t-1}^{t-1}\mathbf{A}' + \mathbf{S}, \tag{3.42}$$

corresponding to the horizontal arrow in Figure 3.4, where sufficient statistics for representing $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ are computed. This is the *a priori* estimate of the state variables.

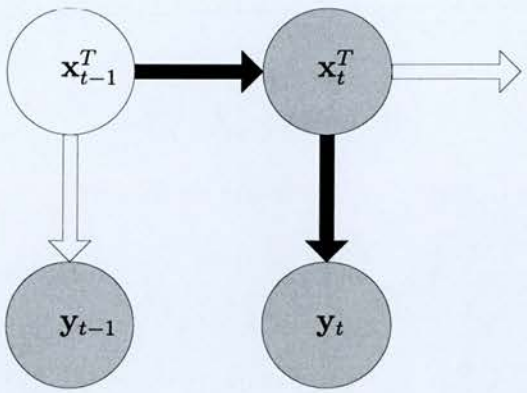


Figure 3.5: Probabilistic Graphical Model fragment illustrating the backwards (Rauch–Tung–Striebel) recursions

In the measurement update step, we obtain the *a posteriori* distribution by taking the next observation, computing the difference between the observation and the predicted observation given the *a priori* state estimate, and then feeding the error term back through a gain matrix $\mathbf{K}_t$, which is computed so as to minimize the state error covariance matrix:

$$\mathbf{K}_t = \mathbf{V}_t^{t-1}\mathbf{W}'(\mathbf{W}\mathbf{V}_t^{t-1}\mathbf{W}' + \mathbf{R})^{-1} \tag{3.43}$$

$$\mathbf{x}_t^t = \mathbf{x}_t^{t-1} + \mathbf{K}_t(\mathbf{y}_t - \mathbf{W}\mathbf{x}_t^{t-1}) \tag{3.44}$$

$$\mathbf{V}_t^t = \mathbf{V}_t^{t-1} - \mathbf{K}_t\mathbf{W}\mathbf{V}_t^{t-1} \tag{3.45}$$

The equations for solving the Smoothing problem compute the distribution of the latent variables given the entire observation sequence, $p(\mathbf{x}_t \mid \{\mathbf{y}\}_1^T)$ and this involves a set of recursions backwards through the data, in order to compute $p(\mathbf{x}_{t-1}|\mathbf{x}_t)$, illustrated schematically in Figure 3.5. They are known collectively as the Rauch–Tung–Striebel recursions.

The derivation of these equations as indicated by the literature [Rauch et al., 1965] is lengthy, but the key step is the marginalization over $\mathbf{x}_{t+1}$ so that once again, d-separation implies that the distribution only depends on $\mathbf{x}_{t+1}$.

The equations operate in an analogous way to the forward equations, first computing a smoother gain matrix $\mathbf{J}_{t-1}$, which is used to update the estimated state and error covariance matrices so they are conditioned on the whole observation sequence.

The gain matrix is given by:

$$\mathbf{J}_{t-1} = \mathbf{V}_{t-1}^{t-1}\mathbf{A}'(\mathbf{V}_t^{t-1})^{-1}. \tag{3.46}$$

The smoother update equations are as follows:

$$\mathbf{x}_{t-1}^T = \mathbf{x}_{t-1}^{t-1} + \mathbf{J}_{t-1}(\mathbf{x}_t^T - \mathbf{A}\mathbf{x}_{t-1}^{t-1}) \tag{3.47}$$

$$\mathbf{V}_{t-1}^T = \mathbf{V}_{t-1}^{t-1} + \mathbf{J}_{t-1}(\mathbf{V}_t^T - \mathbf{V}_t^{t-1})\mathbf{J}'_{t-1} \tag{3.48}$$

The bracketed term in Equation (3.47) is an error term between the current smoothed estimate, and the predicted estimate from the previous time step. This is used in a feedback loop to produce the smoothed estimate for the previous time step. Updating the smoothed variance estimates operates in an analogous way. We also require the smoothed cross-variance terms $V_{t,t-1}$, which are computed from another set of recursions:

$$\mathbf{V}'_{t-1,t-2} = \mathbf{V}_{t-1}^{t-1}\mathbf{J}'_{t-1} + \mathbf{J}_{t-1}(\mathbf{V}'_{t,t-1} - \mathbf{A}\mathbf{V}_{t-1}^{t-1})\mathbf{J}'_{t-2} \tag{3.49}$$

This last recursion is initialized with $\mathbf{V}_{T,T-1}^T = (\mathbf{I} - \mathbf{K}_T\mathbf{W})\mathbf{A}\mathbf{V}_{T-1}^{T-1}$.

3.3.1.2 Explicit incorporation of the constant terms μ and b

As was described in Section 3.3.1, the constant terms for the state evolution $\mathbf{b}$ and the output map $\boldsymbol{\mu}$ can in principle be "absorbed" into the matrices $\mathbf{A}_+$, and $\mathbf{W}_+$, as an extra column, by introducing an extra state variable whose value is always unity. This results in a singular normal distribution for $p(\mathbf{x}_{t+1}^+|\mathbf{x}_t^+)$. This means that the formulation given above cannot be naively implemented in the Kalman Smoother equations given in the previous section. Specifically this will cause a problem in the computation of the smoother gain matrix, in Equation (3.46), because the matrix to be inverted will be singular.

The solution to this problem lies in the fact that in the limiting case, the distribution becomes a product of the standard distribution in the L-dimensional subspace spanned by the latent variables of interest, and a delta function distribution. Hence, when expectations are calculated, which involves integration over the latent variables, only the correct value (when

the auxiliary variable is unity) is picked out, which corresponds to setting $\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{b}$, as desired.

Hence there is no need to evaluate the E-step as an $(L+1)$ dimensional set of equations; we can implicitly take care of the extra state variable by modifying the equations as follows. First, Equation (3.41) becomes:

$$\mathbf{x}_t^{t-1} = \mathbf{A}\mathbf{x}_{t-1}^{t-1} + \mathbf{b}. \tag{3.50}$$

Equation (3.44) becomes:

$$\mathbf{x}_t^t = \mathbf{x}_t^{t-1} + \mathbf{K}_t(\mathbf{y}_t - \boldsymbol{\mu} - \mathbf{W}\mathbf{x}_t^{t-1}). \tag{3.51}$$

Equation (3.47) becomes:

$$\mathbf{x}_{t-1}^T = \mathbf{x}_{t-1}^{t-1} + \mathbf{J}_{t-1}(\mathbf{x}_t^T - \mathbf{A}\mathbf{x}_{t-1}^{t-1} - \mathbf{b}). \tag{3.52}$$

Finally, we note that as we have actually only explicitly computed the expectations of the original state vectors, rather than the augmented state vectors, we need to construct the required matrices in the M-step updates from the computed quantities, which is straightforward. Hence (3.31) becomes:

$$\mathbf{A}_+^{\text{new}} = \left[\sum_{t=1}^{T-1} \begin{pmatrix} \langle \mathbf{x}_{t+1}\mathbf{x}_t' \rangle & \langle \mathbf{x}_{t+1} \rangle \\ \langle \mathbf{x}_t \rangle & 1 \end{pmatrix}\right] \left[\sum_{t=1}^{T-1} \begin{pmatrix} \langle \mathbf{x}_t\mathbf{x}_t' \rangle & \langle \mathbf{x}_t \rangle \\ \langle \mathbf{x}_t \rangle & 1 \end{pmatrix}\right]^{-1}. \tag{3.53}$$

The other update equations in the M step may be obtained in a similar fashion.

3.3.1.3 Extending the model for higher order Kalman Filter Dynamics

If desired, higher order Kalman Filter Dynamics can be incorporated, while still retaining the PCA type visualization. Normally, in order to increase the order of the Kalman filter, one would simply increase the dimension of state space, by increasing the size of the state evolution matrices. However, in the case of visualization, we wish to retain the same dimension for visualization space. We wish, therefore to have a set of linear dynamical equations of the following form:

$$\mathbf{x}_{t+1} = \mathbf{A}_1\mathbf{x}_t + \mathbf{A}_2\mathbf{x}_{t-1} + \mathbf{b} + \mathbf{r} \tag{3.54}$$

$$\mathbf{y}_t = \mathbf{W}\mathbf{x}_t + \mathbf{v}. \tag{3.55}$$

This can be incorporated into the existing algorithm in a similar manner to incorporating the constant drift term by augmenting the state vector to include the previous state vector. Thus the new augmented state vector is $\mathbf{x}_t^{++} = [\mathbf{x}_t', \mathbf{x}_{t-1}', 1]'$, and the new state dynamics equations are:

$$\begin{pmatrix} \mathbf{x}_{t+1} \\ \mathbf{x}_t \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{A}_1 & \mathbf{A}_2 & \mathbf{b} \\ \mathbf{I} & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} \mathbf{x}_t \\ \mathbf{x}_{t-1} \\ 1 \end{pmatrix} + \begin{pmatrix} \mathbf{r} \\ 0 \\ 0 \end{pmatrix} \tag{3.56}$$

.

The output mapping matrix $\mathbf{W}$ is now a $D \times (2L + 1)$ matrix, with columns $L + 1 \ldots 2L$ set to zero, and column $L + 1$ set to $\boldsymbol{\mu}$.

Once again, in order to perform the Kalman filtering/smoothing, we need to factorize the distribution into a product of an L dimensional multivariate normal distribution and delta function distributions:

$$p(\mathbf{x}_{t+1}^{++}|\mathbf{x}_t^{++}) = (2\pi)^{-L/2}|S|^{-1/2}$$
$$\exp\left\{-\frac{1}{2}(\mathbf{x}_{t+1} - \mathbf{A}_1\mathbf{x}_t - \mathbf{A}_2\mathbf{x}_{t-1} - \mathbf{b})'\mathbf{S}^{-1}(\mathbf{x}_{t+1} - \mathbf{A}_1\mathbf{x}_n - \mathbf{A}_2\mathbf{x}_{t-1} - \mathbf{b})\right\}$$
$$\boldsymbol{\delta}(\|\mathbf{x}_n - \mathbf{x}_{n-1}\|)\boldsymbol{\delta}(x_n^{(L+1)} - 1), \tag{3.57}$$

and so the same argument about the computation of expectations applies, making the extension of the algorithm straightforward.

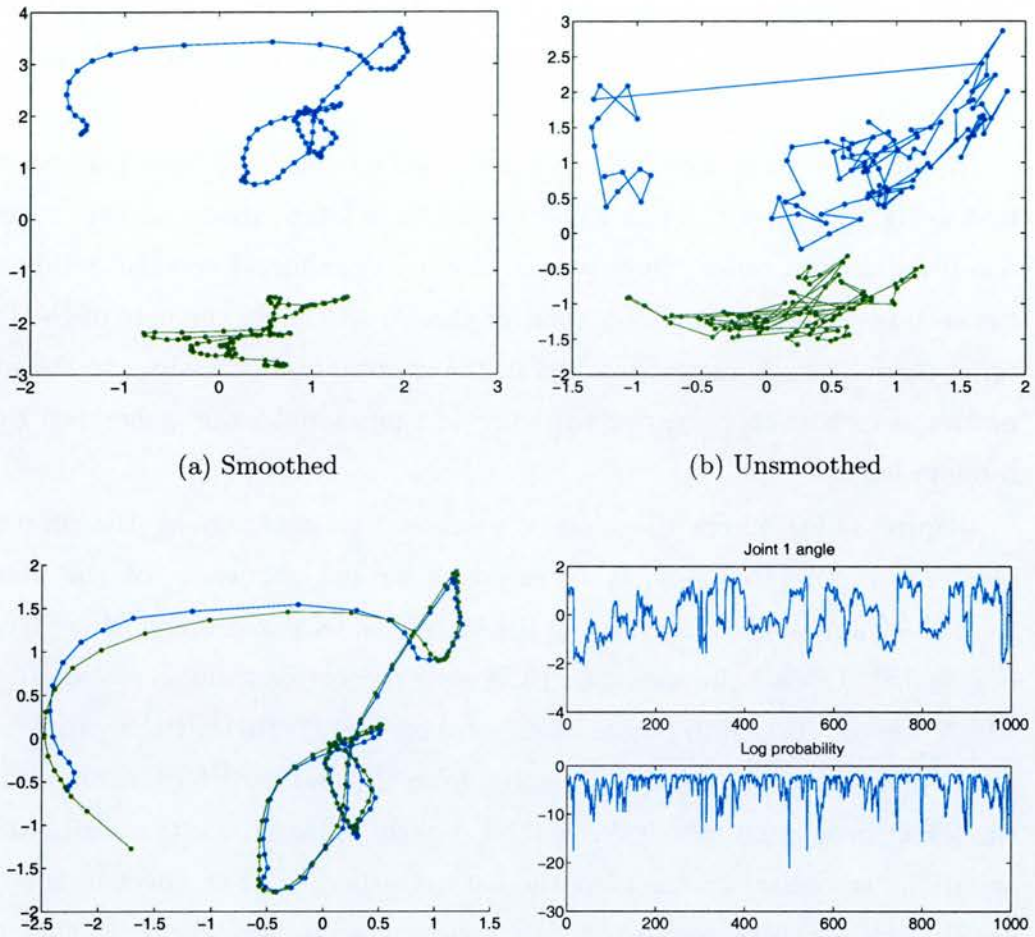
3.4 Demonstration with toy data

We now demonstrate the kind of visualization that can be achieved with PPCA through time with a toy data problem, namely the "switching state robot" example. This is described in detail in Appendix A.2. Briefly the data set generated is four dimensional, consisting of the joint angles and Cartesian position of a two link robot manipulator, as the joint angles are

driven on a Gaussian random walk. Additionally, occasional large transitions are artificially introduced into the data by changing the joint angles between two possible solutions for a given (x, y) location in Cartesian space. A data set of 100 sequences of 100 vectors was generated, and used to train the models.

Results are illustrated in Figure 3.6. Two models have been trained, the first using first order Kalman Filter dynamics in latent space, and the second one using second order. In this case, the final likelihood was little different between the two (marginally higher for second order). In the case of the first order model, the $\mathbf{A}$ matrix learned in the process was very close to the unit matrix, which is to be expected, since the input data was generated by a random walk.

Figure 3.6(a) shows the kind of visualization achieved by the method, plotting out the estimated state variables for two sequences of 100 points from the Kalman Smoother, using the learnt model parameters. By contrast Figure 3.6(b) shows the straight PPCA style projection using Equation (3.8), which does not take into account temporal context. Note that this projection is not precisely the same as obtainable from the standard PPCA algorithm; the learnt projection matrix $\mathbf{W}$ will be slightly different, as its re-estimation equations are based on the smoothed state estimates. The effect of the linear dynamics can be seen clearly by comparing the two traces, in that the temporal plot is much smoother, and that furthermore, when a sudden large transition is made (which of course is not well-modelled by a linear dynamical system), there are several intermediate points as the smoothed trace moves across latent space, lagging behind the un-smoothed data, which makes the transition in one step. In Figure 3.6(c), a comparison is made between the first order model (blue trace) and the second order model (green trace), for the first sequence of 100 points. Although the results are broadly similar, it appears that the second order model has produced a greater degree of smoothing, and may have a slightly longer time constant. It is, of course, a matter of debate whether this is an advantage or not. The greater the degree of smoothing, the more information is discarded. There will always be a trade-off between the clarity of visualization (achieved by smoothing which shows the gross trends), and accuracy of representation. In Figure 3.6(d), we



(a) Smoothed

(b) Unsmoothed

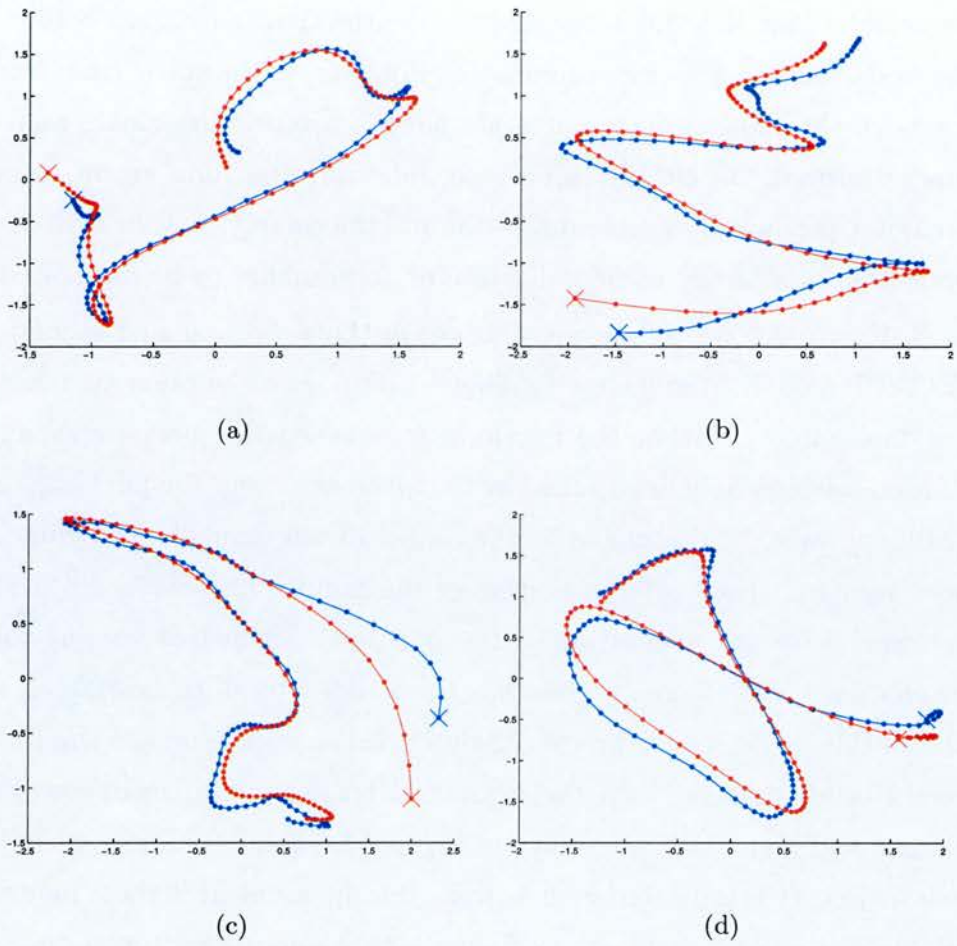
(c) 1st and 2nd Order

(d) Log probability

Figure 3.6: PPCA Through Time with the switching-state robot data. The estimated state variables from the Kalman Smoother for two trajectories are shown in (a), whereas (b) shows the corresponding unsmoothed traces obtained simply by projecting the data using the output matrix from the Kalman Filter equations. In (c) the smoothed latent variables for the first order model (blue) and the second order (green) are compared. Finally (d) shows one of the input variables (joint 1 angle) correlated with the probability for the first order model of the observed data given the previous observations in the sequence. Sharp drops in the probability tend to correspond to large transitions in the data.

compute $\log p(\mathbf{y}_n|\{\mathbf{y}_1, \ldots \mathbf{y}_{n-1}\})$, i.e. the probability of the current data point given the model and the previous observation sequence. This is displayed in parallel with the value of the joint 1 angle. The graphs show that the sudden transitions in the value of the variable usually correspond to drops in the probability, indicating that the PPCA through time dynamical model has detected an unusual event (i.e. one which it was not capable of modelling correctly). This illustrates the discussion attendant on Figure 3.1 — when the model is poor, the prior dominates. However, in the static case, this does not help; the plotted data points are shrunk towards the mean, and hence appear normal. In the dynamic case, however, the time lag in the linear dynamics produces a visible indication of abnormality, and in addition, the model allows a probabilistic indication of abnormality to be calculated.

In Figure 3.7 we compare visualizations between first and second order PPCATT models from a slightly different data set. The previous robot data was "noise-free", in that the relationship between the joint angles and the position was precisely determined by the kinematics, but the joint angles were a random walk. In this example, the random walk inputs to the joint angles have been smoothed using a first order linear filter ($\hat{x}_n = 0.1x_n + 0.9x_{n-1}$). However, after the generation of the position coordinates via the forward kinematics, extra Gaussian noise has been added to all the variables, so the relationship is no longer precise. Side by side comparisons of the first and second order models, with the smoothed trajectory superimposed on the unsmoothed trajectory, are shown in Figures 3.8 and 3.9. The start point for each trajectory is indicated with a cross. It is apparent from these figures that PPCATT is again capable of producing highly smoothed trajectories. There will in practice always be a trade-off as to whether this is a desirable property or not. If the data being analysed is excessively noisy, then the first order model will be more sensitive to noise than the second order. However, as a result of the greater degree of smoothing, the second order model is slower to respond to general changes in the data, that are due to underlying trends, rather than to noise. This effect is apparent in the comparison between Figures 3.9(a) and 3.9(b), where the start of the trajectory is seen on the right of the plots around half way up. Both the smoothed trajectories exhibit start-up transients, where the path goes from the initial state vector $\mathbf{x}_0$, to follow



(a) (b)

(c) (d)

Figure 3.7: Comparison between first order (blue) and second order (red) Kalman Filter dymamics for 2-D robot with smooth trajectory and added measurement noise.

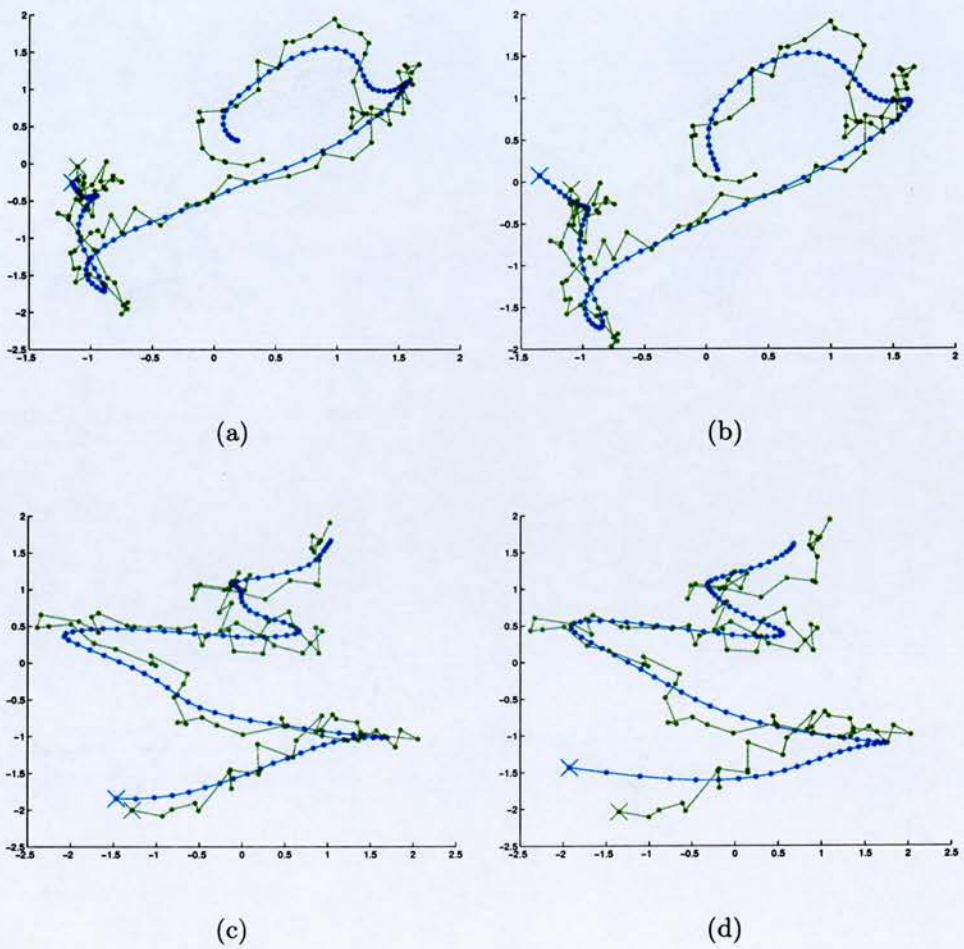


Figure 3.8: Side by side comparison of the first and second order models for the first two sequences from Figure 3.7. Smoothed traces from Kalman-Smoother are shown in blue, and the unfiltered projections in green. The results from first order systems are shown in (a) and (c), and the corresponding second order results are shown in (b) and (d). The start point of each trajectory is marked with an cross.

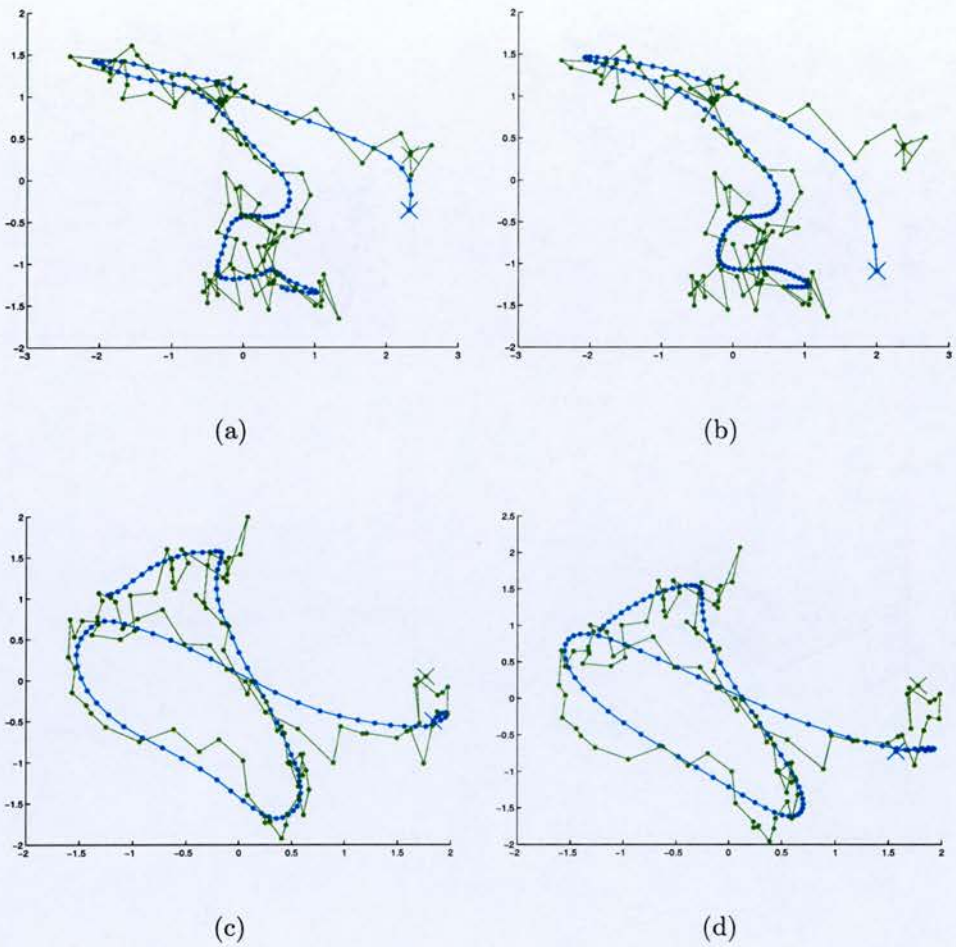


Figure 3.9: Side by side comparison of the first and second order models for the last two sequences from Figure 3.7. Smoothed traces from Kalman-Smoother are shown in blue, and the unfiltered projections in green. The results from first order systems are shown in (a) and (c), and the corresponding second order results are shown in (b) and (d). The start point of each trajectory is marked with an cross.

the data. It is clear from the diagrams that in the second order case, this transient takes longer to settle, in that it takes 8 samples before it appears to be tracking the trend of the unsmoothed data. In the case of the first order model, the corresponding time is only 4 samples.

3.5 Discussion

In this chapter, we have shown how methods for probabilistic learning of linear dynamical systems can be applied to visualization techniques, where the state space of the linear dynamical system is used to provide the visualization. The output model leads to a visualization technique that is a temporal version of Principal Components Analysis.

We have further demonstrated that higher order linear dynamics can be incorporated into the visualization space, by adopting a constrained form of the state evolution equations.

The technique behaves much as we expect; it is able to filter out noise, which by definition has no temporal correlation, while still being able to track the trends of the data. A key idea is the property of shrinking the posterior distribution towards the prior, illustrated for the static case in Figure 3.1. In the static case, this is a feature that has no use at all, but in the temporal case, it leads to the ability to smooth out noise (because the projection of the new point is shrunk towards the posterior mean of the previous point), and furthermore, it can show when an unusual event (that is not modelled well by linear dynamics) has taken place, because there will be a temporal lag in the plotted data. Where this is the case, it is helpful to plot the smoothed and unsmoothed data together.

The technique has two principal limitations. The first limitation of the model is the restriction to a linear mapping. This means that it will be poor at estimating probabilities for systems which live on a non-linear manifold. Although the PCA style projection will produce a recognisable visualization, the probability will vary according to how far out-of-plane the data point happens to be. However, this would not necessarily be a reflection on the data showing abnormal tendencies, but the inability of the model to capture all the features of the data.

The second limitation is that it is limited only to linear dynamics, and the dynamical behaviour is the same over all of latent space. This would not, therefore, be appropriate for a dynamical system that showed different behaviour in different regions, such as the Lorenz Attractor, which will be

modelled in detail in the next chapter. Figure 3.10 illustrates the type of visualization of this data set achievable with PPCATT:

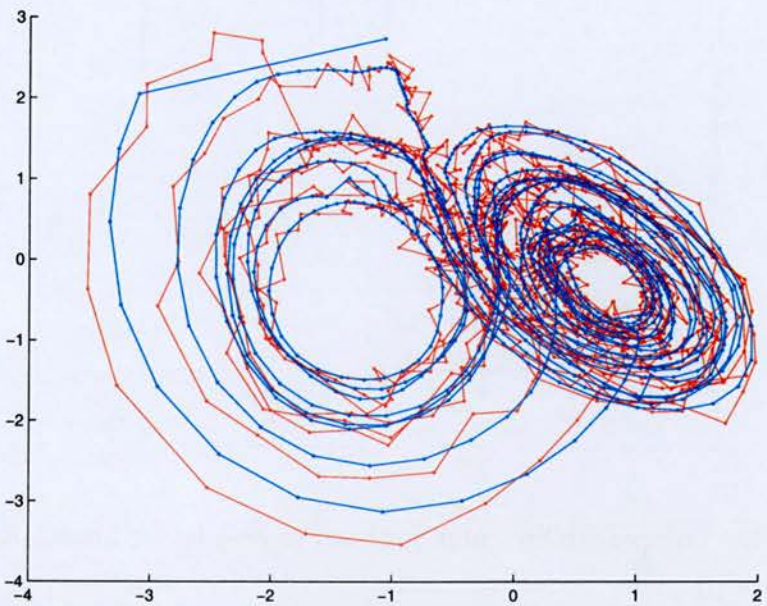


Figure 3.10: PPCATT visualization of data generated by the Lorenz Attractor dynamical system, with added noise.

The data set (which is described in detail in Appendix A), is a three-dimensional dynamical system, consisting of two rotating "eddies" that spiral outwards, one clockwise and the other anticlockwise. Occasionally, there are fast transitions from one eddy to the other. It is clear that the dynamical behaviour varies across the data space, and hence a single global dynamical system will not be well suited to forming a probabilistic model. In this case, the right-hand "eddy" is better modelled than the left-hand one, and this is reflected in the plot of log probability of each observed data point for the points plotted in Figure 3.11.

Here, it is clear first of all that the probability appears to vary in cycles, and this will be due to the cyclic variation of the data. Since the two eddies lie in different planes, it is impossible to place a linear plane through both of them. Therefore the out-of-plane distance will also vary in cycles, resulting in the probability varying in a similar fashion. It appears that the projection plane has aligned better with the right hand eddy of Figure 3.10, as the variations in probability are less violent for the first few hundred data sam-

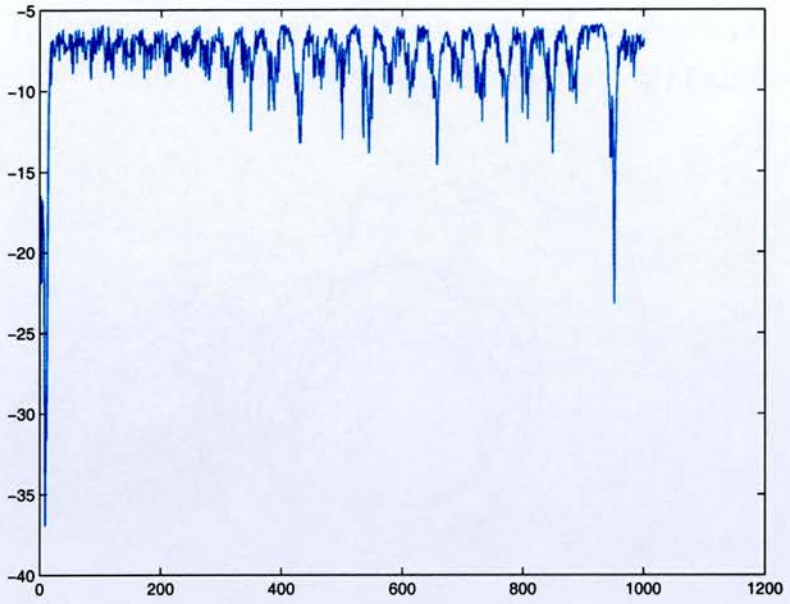


Figure 3.11: Log probability curve for each sample in the Lorenz Attractor data of Figure 3.10.

ples. The more violent variations may also be a result of an inappropriate dynamical model for this half of the diagram.

Both the above limitations are overcome in the technique introduced in the next chapter, which allows a non-linear manifold to be fitted to the data, and which also is able to model (in principle) arbitrary non-linear dynamical behaviour.

Chapter 4

Generative Topographical Mapping through Time

In this chapter, we introduce a non-linear visualization technique for time-dependent data, based on the Generative Topographical Mapping (GTM) technique of [Bishop et al., 1998b] . GTM is a latent variable model, for providing an L dimensional visualization of data in a D dimensional data space. It is a constrained Gaussian Mixtures model, where the centres of the Gaussian components are constrained to lie on a non-linear L-dimensional manifold in data space. Each mixture component is also constrained to have the same variance. The non-linear manifold is a functional mapping whose parameters can be learned via a maximum likelihood EM algorithm.

We shall develop a time-dependent version of the GTM by treating the Mixture components of the standard GTM model as the hidden states of a Hidden Markov Model (HMM), and adapting the E step of the EM algorithm to compute responsibilities for the components that take temporal context into account. The Baum-Welch re-estimation procedure will then be used to re-estimate the parameters of the HMM, and the standard GTM M step re-estimation equations will be used for the GTM parameters, using modified responsibilities that have been computed by the HMM forward-backward algorithm

The layout of the chapter is as follows:

- In Section 4.1, we shall review the GTM algorithm for the time indepen-

dent case. In particular, various computational issues will be addressed that have been observed during the development of the GTM through time algorithm, leading to a new sparse formulation of GTM that could in principle be used to scale up GTM to very large models.

- In Section 4.2 we shall review the Hidden Markov Model formulation in the context of GTM through time, and present present the basic GTM through Time algorithm. We review techniques for reducing the number of parameters in the model. We introduce a simple scale-up procedure that allows successively larger models to be trained, using a smaller previously trained model as a starting point.

- Finally in Section 4.3, we demonstrate the algorithm and postprocessing options on some simple example data sets.

4.1 GTM for the static case

GTM is a model for representing the probability density of a data set in D dimensions, in terms of a lower dimensional representation of latent variables. Typically, the dimension L of latent variable space is 2, and hence the method is used to visualize data.

The probability density model is produced by considering a smooth non-linear mapping $\mathbf{g}(\mathbf{x}; \mathbf{W})$, where $\mathbf{W}$ is a set of parameters that determine the shape of the mapping, $\mathbf{g}$ is a vector in D dimensions, and $\mathbf{x}$ is a vector in L dimensions.

The probability density is represented as a mixture of spherical Gaussian functions, each with an equal mixing coefficient[1]. GTM is shown schematically in Fig 4.1, where the dimension of data space is taken to be 3, for illustrative purposes.

4.1.1 Review of basic algorithm

First, we review the basic algorithm for GTM for the case of i.i.d data (i.e. not taking temporal correlation into account). Vectors in data space are

[1]This restriction is not inherent in the model itself, but has generally been used for computational simplicity, and to reduce the number of user-definable parameters.

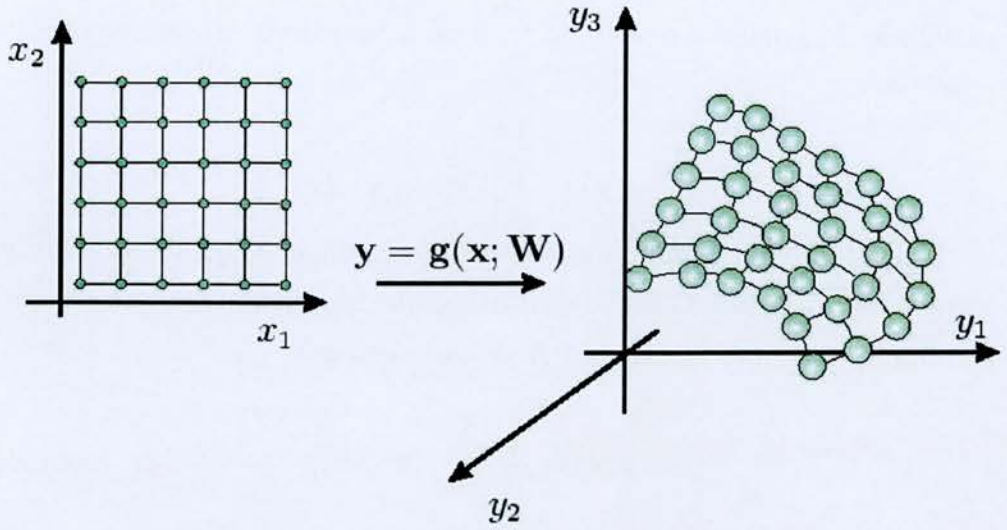


Figure 4.1: GTM illustrated as a constrained Gaussian Mixtures Model, where the centres of each Gaussian component are constrained to line on a 2-dimensional non-linear manifold in data space, and also to have the same variance. The non-linear mapping $\mathbf{g}(\mathbf{x}; \mathbf{W})$ from latent space $\{x_1, x_2\}$ to data space $\{y_1, y_2, y_3\}$, maps a regular array of grid points (shown by small green disks), onto a smooth manifold in data space, whose locations are the centres of the Gaussian Mixture components.

denoted by the symbol $\mathbf{y}$ where $\mathbf{y} = (y_1, y_2, \ldots, y_D)$. Latent space vectors $\mathbf{x}$ are similarly represented $\mathbf{x} = (x_1, \ldots, x_L)$.

For convenience of the calculation, the non-linear mapping is chosen to be a linear combination of M fixed basis functions:

$$\mathbf{g}(\mathbf{x}; \mathbf{W}) = \mathbf{W}\boldsymbol{\phi}(\mathbf{x}). \tag{4.1}$$

In principle, any non-linear mapping capable of universal function approximation can be used, such as the Multi-Layer Perceptron (MLP). However this form is particularly straightforward for use in an E-M maximum likelihood algorithm, because the M-step reduces to a single matrix inversion, as opposed to having to use a full non-linear optimization that would be necessary with an MLP.

As in the previous chapter, the latent variable probability density model is defined in terms of a prior distribution $p(\mathbf{x})$ over latent space, and a con-

ditional distribution $p(\mathbf{y}|\mathbf{x})$ in data space, which is conditioned on the latent variables. The density model can then be obtained by integrating over latent space:

$$p(\mathbf{y}) = \int_{\mathbf{x}} p(\mathbf{y}|\mathbf{x})p(\mathbf{x})\ \mathrm{d}\mathbf{x}. \tag{4.2}$$

For a non-linear mapping, this integral will in general be analytically intractable, so in the GTM algorithm, a specific form of the prior is chosen, consisting of a superposition of K delta functions:

$$p(\mathbf{x}) = \frac{1}{K}\sum_{k=1}^{K}\delta(\mathbf{x} - \mathbf{x}_k), \tag{4.3}$$

where $\{\mathbf{x}_k\}$ corresponds to a set of points in latent space that are analogous to the "feature space" nodes of the SOM. This choice of prior results in a model that bears some similiarity with the SOM, but also allows the integral in Equation (4.2) to be performed by summing the values of $p(\mathbf{y}|\mathbf{x})$ over the points $\{\mathbf{x}_k\}$:

$$p(\mathbf{y}) = \frac{1}{K}\sum_{k=1}^{K}p(\mathbf{y}|\mathbf{x}_k) \tag{4.4}$$

Finally, we choose the conditional density $p(\mathbf{y}|\mathbf{x})$ to be a radially symmetric Gaussian, with mean $\mathbf{y}$ given by the non-linear mapping $\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{W})$ and inverse variance β:

$$p(\mathbf{y}|\mathbf{x}) = \left(\frac{\beta}{2\pi}\right)^{D/2}\exp\left\{-\frac{\beta}{2}\|\mathbf{y} - \mathbf{g}(\mathbf{x}, \mathbf{W})\|^2\right\} \tag{4.5}$$

The adjustable parameters of the GTM model $\{\mathbf{W}, \beta\}$ are then obtained by maximum likelihood, using an EM algorithm. The M step of this algorithm is obtained by computing the derivatives of the log likelihood function with respect to the elements of $\mathbf{W}$ and β, and equating to zero. The log likelihood is defined for a set of N data points $\{\mathbf{y}^1, \mathbf{y}^2, \ldots \mathbf{y}^N\}$:

$$\mathcal{L}(\mathbf{W}, \beta) = \sum_{n=1}^{N}\ln\left\{\frac{1}{K}\sum_{i=1}^{K}p(\mathbf{y}^n|\mathbf{x}^i, \mathbf{W}, \beta)\right\} \tag{4.6}$$

This leads to the solution for $\mathbf{W}$:

$$\mathbf{W}^T = (\mathbf{\Phi}'\mathbf{G}\mathbf{\Phi})^{-1}\mathbf{\Phi}'\mathbf{R}\mathbf{Y} \tag{4.7}$$

and for β:

$$\frac{1}{\beta} = \frac{1}{ND}\sum_{k=1}^{K}\sum_{n=1}^{N} R_{kn}(\mathbf{W}, \beta)\|\mathbf{g}(\mathbf{x}^k; \mathbf{W}) - \mathbf{y}^n\|^2 \tag{4.8}$$

where $\mathbf{\Phi}$ is a $K \times M$ matrix with elements $\phi_{ij} = \boldsymbol{\phi}_j(\mathbf{x}^i)$, $\mathbf{Y}$ is an $N \times D$ matrix whose rows consist of the data vectors $\mathbf{y}$, and $\mathbf{R}$ is a $K \times N$ matrix of responsibilities of each of the Gaussian mixture components for each of the data points. The elements R_{kn} are given by:

$$\mathbf{R}_{kn}(\mathbf{W}, \beta) = \frac{p(\mathbf{y}^n|\mathbf{x}^k)}{\sum_{k'=1}^{K}(\mathbf{y}^n|\mathbf{x}^{k'})}, \tag{4.9}$$

The term $\mathbf{G}$ in equation 4.7 is a $K \times K$ diagonal matrix whose elements are the sums of the rows of the responsibility matrix. So, Equation (4.9) constitutes the E-step of the algorithm, where the expected distribution of latent space variables is computed. Sufficient statistics for this distribution are stored for each point as a single column in the matrix of responsibilities $\mathbf{R}$. Equations (4.7) and (4.8) constitute the M step, where the parameters $\{\mathbf{W}, \beta\}$ are adjusted to maximize the likelihood, given the expected value of the posterior distribution. These equations may be derived by substituting Equation (4.5) into Equation (4.6), differentiating and equating to zero. The detailed derivation of these re-estimation formulae is given in [Svensén, 1998]

.

So far, nothing has been said about the choice of basis functions for the GTM model. The standard choice adopted in [Bishop et al., 1997b], [Bishop et al., 1998a]and [Svensén, 1998] has been to use a combination of constant, linear and non-linear basis functions. There are L linear basis functions, corresponding to each of the directions in latent space, in order to capture the linear trends in the data set. The single constant basis function is a constant offset. Finally, there are $M - L - 1$ non-linear basis functions, which are radially symmetric Gaussian functions of the form:

$$\phi_j(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \boldsymbol{\mu}_j\|^2}{2\sigma^2}\right) \tag{4.10}$$

where σ governs the smoothness of mapping achieved, and the $\boldsymbol{\mu}_j$ are generally chosen to be a square grid in latent space. In all work carried out so far, it has been normal for there to be at least twice as many sample points in each

dimension as there are basis functions, and for the widths of the non-linear basis functions to cover at least twice the separation between neighbouring $\boldsymbol{\mu}$ positions in latent space. Smaller values of σ coupled with a large value of M will lead to a more flexible mapping, and hence a better density model. However, as noted in the abstract of [Svensén, 1998] , the added complexity can also lead to a more convoluted and twisted map, making the interpretation of the results more difficult. Thus there is a trade-off between the desire for a good probabilistic model and the need for an easy visualization. This was noted in the above reference as a fundamental limitation of the GTM model.

This motivates a more subjective approach to model order selection than would be the case, for example when developing a probabilistic model for pattern classification, such as hand-written digit recognition. In such cases, we could compare the effectiveness of different models by comparing directly the data likelihood on the test set, and choosing the model with the highest likelihood. However, in the case of a model such as GTM, the higher likelihood model, even if it can be demonstrated to perform better on a test set, does not necessarily provide the best visualization, because the complexity of the map will make interpretation of the visualization obtained more difficult. As there is not, in this case, an objective measure of the merit of a particular visualization, we have to use subjective measures such as the general smoothness of the manifold obtained.

4.1.2 Sparse representation of the GTM mapping

In this section, we introduce a slightly modified version of the GTM mapping that has an important effect as far as the scalability of the algorithm is concerned.

GTM was proposed as a principled alternative to the Self Organising Map, because it is based on a probabilistic formalism. However, there is an associated computational cost that seems to restrict its use to small or moderate sized grids. As can be seen from Equation (4.7) the update formula for the output weights matrix $\mathbf{W}$ involves the solution of a simultaneous set of linear equations, with computational complexity $\mathcal{O}(M^3)$. By contrast, the

batch update rule of the SOM of Equation (2.19) only has complexity $\mathcal{O}(M^2)$. In fact, the most computationally demanding part is not the linear equation solution, but the computation of the matrix to be inverted, $\mathbf{\Phi}'\mathbf{G}\mathbf{\Phi}$. It is normal in GTM to have the grid separation set to half the basis function centre separation, meaning that $K = 4M$, and hence $4M^3$ floating point operations will be required to compute the matrix to invert[2]. This seems to place serious restrictions on the scalability of GTM. By contrast, the SOM has been demonstrated on some extremely large applications, for example in [Kohonen et al., 2000], where the final map had 1,002,240 nodes.

Naturally, if one simply wishes to increase the *resolution* of the grid-style mapping that emerges from GTM, then all that is needed is to increase K, without increasing the number M of basis functions. Then the scaling of the computational effort is only linear with respect to K. However, although this will increase the resolution of the mapping obtained, it will not increase the complexity of the representation, and would be analogous to a SOM that had been trained down to a large value of final neighbourhood. In order to explore in more detail the features of the data set (as opposed to treating them as noise), we must also increase M, and this leads to the undesirable cubic scaling law.

The solution is to exploit the localized nature of the Gaussian basis functions, by approximating the $\mathbf{\Phi}$ matrix with a sparse representation of itself.

Figure 4.2, shows a pseudo-colour plot of the non-linear submatrix of the $\mathbf{\Phi}'\mathbf{\Phi}$ matrix for a moderately large GTM model (3600 mixture components with 900 non-linear basis functions). Note that the majority of the elements in the matrix have a very small value, and are indicated by black on the pseudo-colour plot. The matrix itself has a banded structure. Since we are not restricted to a strict Gaussian for the non-linear basis functions it is perfectly permissible to choose a basis function that truncates to zero when the separation from the centre exceeds a certain value. Indeed, we note from [Kohonen et al., 2000], that when Gaussian neighbourhood functions are used in the SOM algorithm, that this practice is also adopted in the

[2]Note that each term in the summation of a matrix–matrix multiplication requires an addition and a multiplication, but that the matrix required is symmetric, so only the upper or lower triangle need to be computed.

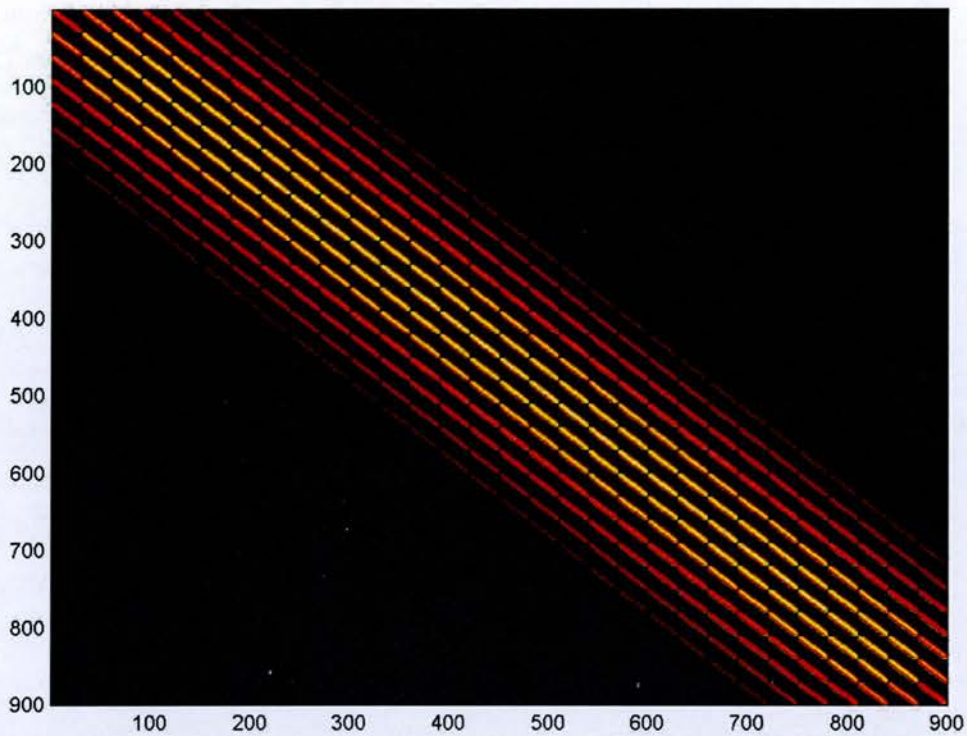


Figure 4.2: Pseudocolour plot of the non-linear submatrix of $\mathbf{\Phi}'\mathbf{\Phi}$ matrix for a GTM model with a 60×60 grid of sample points and a 30×30 grid of non-linear basis functions.

neighbourhood update procedure, reducing the scaling law from Equation (2.19) from quadratic to linear. Similarly, the scaling law for GTM will also be linear when basis function truncation is taken into account. We now illustrate the scaling properties of the law in more detail.

The computational complexity of calculating the term $\mathbf{A} = \mathbf{\Phi}'\mathbf{G}\mathbf{\Phi}$ may be estimated by considering Figure 4.3, which is a schematic representation of latent space, using a rectangular lattice. The M basis functions are centred on the green small circles, and the K delta functions of the GTM prior are centred on the grid intersections in the diagram. The grey circles represent the radius of truncation of each of the basis functions ϕ_i and ϕ_j. It is clear that as the basis functions are truncated outside the circles, then the term A_{ij} can only be non-zero if the circles intersect, and the intersection hap-

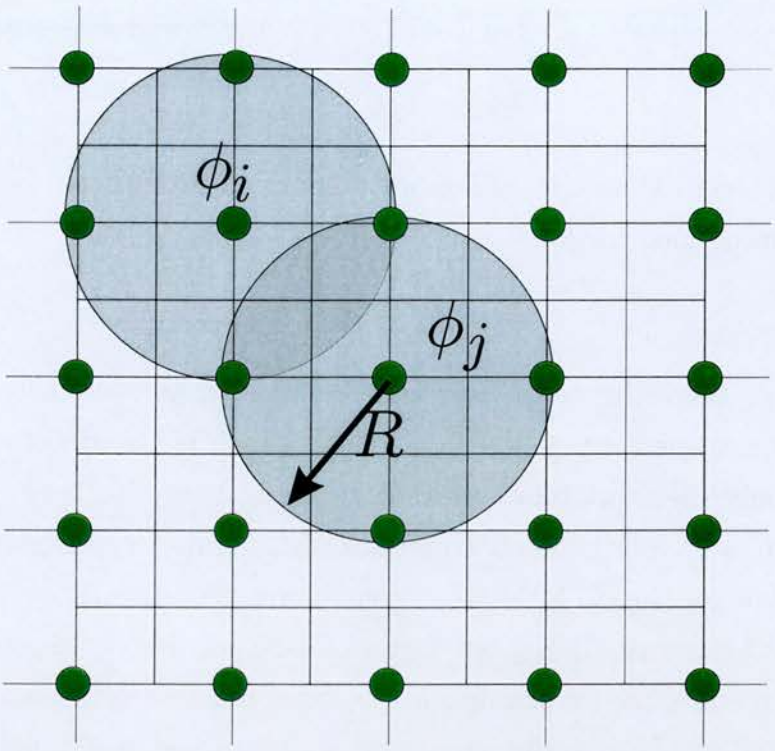


Figure 4.3: Computation of the matrix element $\mathbf{\Phi}'\mathbf{G}\mathbf{\Phi}$ for the GTM M step, with truncated basis functions. The two grey circles represent the radius R of truncation of the two basis functions ϕ_i and ϕ_j. Element (i, j) of the resultant matrix will be non-zero only if the circles intersect

pens to contain one or more grid points, because the value of the element is $\sum_{k=1}^{K} \phi_{ik}\phi_{jk}$, and the basis function is only non-zero inside the circles. From this, we can deduce that column i of the matrix will only contain as many non-zero elements as there are basis function centres inside a circle of radius $2R$, which, for a large radius, we can approximate as $\rho\pi(2R)^2$, where ρ is the density of basis function centres per unit area. Similarly, we compute that the *maximum* number of floating point multiplications will be when the two circles are superimposed and $i = j$, which comes to the number of latent space sample points inside the circle of radius R, which is approximated by $4\rho\pi R^2$ (noting that the density of sample points is four times the density of basis function centres).

Hence, if we define the sample point separation to be unity, making $\rho = 1/4$, we arrive at the conclusion that the number of non-zeros in the matrix

whose basis functions are truncated outside a radius of R is given by:

$$\text{nnz}(\mathbf{\Phi}'\mathbf{G}\mathbf{\Phi}) = M\pi R^2, \tag{4.11}$$

and that the total number of floating point operations (flops) required obeys the following inequality:

$$\text{nflops}(\mathbf{\Phi}^T\mathbf{G}\mathbf{\Phi}) < M\pi^2 R^4. \tag{4.12}$$

In fact the estimate in Equation (4.12) is an over-estimate, because it assumes maximum overlap of the circles for each non-zero element, whereas in fact only the overlap between the circles will vary between zero and the maximum amount. However equations (4.11) and (4.12) illustrate the true scaling law for the GTM M step calculations when truncation is applied to the basis functions. For a given degree of local curvature of the map, we wish to keep R constant (for example at two basis function grid separations), and increase M and keep to the relation $K = 4M$. In this case, it is seen that the complexity of calculating the matrix to invert, and the number of non-zeros in it scales linearly with M.

It is harder to estimate the true time for the solution of the linear equations, which will involve a Cholesky decomposition of the matrix. For the dense case, this, too is an $\mathcal{O}(M^3)$ operation. The number of operations required generally for a sparse system varies linearly with the number of non-zeros in the matrix, but is also dependent on the matrix structure, which can cause different amounts of fill-in in the matrix factorization. However, we can obtain a "worst case" estimate by noting that the matrix in Figure 4.2 has a banded structure (with sparsity within the band). If we treat it as a dense band structure with bandwidth N_b, then the number of operations required to perform the matrix factorization is given as $2MN_b^2$, ([Dongarra et al., 1979]), and hence is also linear in M. In practice, as noted above, the solve time is negligible compared to the time need to compute the matrix itself, and scales linearly as well.

Finally, we should note that if the linear and constant basis functions are included into the equations to solve, that the matrix no longer has the band-diagonal structure. In practice, given a reasonable sparse symmetric matrix solver, this does not pose any problem at all. However, we can demonstrate

rigorously that the scaling law is not impaired by expressing the new set of equations to solve as an extended set of $M+3$ equations in $M+3$ unknowns (assuming that $L = 2$ and we have an extra constant basis function:

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}' & \mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}, \tag{4.13}$$

where $\mathbf{A}$ is the $M \times M$ sparse matrix we have considered previously, $\mathbf{B}$ is an $M \times 3$ matrix of cross-terms between the linear and constant basis functions and the non-linear basis functions, and $\mathbf{C}$ is a 3×3 matrix. The solution of such a set of equations can be determined once the LU factorization of the matrix $\mathbf{A}$ has been obtained, by forming the Schur compliment matrix $\mathbf{S} = \mathbf{C} - \mathbf{B}'\mathbf{A}^{-1}\mathbf{B}$. It is then a standard procedure to compute the full solution, and the extra steps, including computing the Schur compliment, all scale linearly with M.

We conclude, therefore, that the scaling law for GTM is the same as that for SOM.

4.1.2.1 Specific Example of speed up obtained with sparsity

In the very large SOM network described in [Kohonen et al., 2000], the analogue of K is the number of code book vectors in the map, or 1,002,240. Hence M would be $250,560$, and the number of floating point operations required to evaluate $\mathbf{\Phi}'\mathbf{G}\mathbf{\Phi}$ would be 6.29×10^{16}, which, on a 1 gigaflop machine working at peak rate would take 728 days (and this is for each iteration of the EM algorithm). Additionally the storage required for the matrix would be 5×10^{11} bytes.

We now estimate the time required for the computation using a sparse representation, and allow the sparsity to arise "naturally" because of the truncation of small numbers to zero in finite precision arithmetic. The smallest floating point number represented in the IEEE floating point standard for double precision arithmetic has a value of 2.2251×10^{-308} (a quantity defined in Matlab as `realmin`). Hence even if we do not artificially truncate the basis functions, we will find that they are truncated by the rounding error of the computer if:

$$\exp\left(-\frac{1}{2\sigma^2}R^2\right) < 2.2251 \times 10^{-308}. \tag{4.14}$$

But the limit is more severe than this, because if the values of the basis functions in the intersection of the circles in Figure 4.3 are all less than the square root of `realmin`, then their product will round to zero as well. Hence, taking logs, we determine that for non-zero elements to occur, the radius R must satisfy

$$\frac{1}{2\sigma^2}R^2 < 354.19 \tag{4.15}$$

In the million node network of [Kohonen et al., 2000], the radius of the neighbourhood function is stated as being at nine grid point separations. Noting that this is equivalent to 4.5 basis function separations in the arrangement of Figure 4.3, we can construct an equivalent GTM formulation, where three standard deviations of the Gaussian correspond to 4.5 basis function separations, and hence 9 grid nodes. Hence $\sigma = 1.5$, and we compute the maximum R to be 39.92. We thus see, that in a grid of side 1000, that the matrix is very sparse indeed, simply due to round-off error. The number of non-zero elements will be 5.01×10^6 (meaning that the storage requirements are straightforward for modern machines). The number of floating point operations is estimated at 6.3×10^{12}, requiring 104 minutes on a 1 Gigaflop machine, assuming peak rate is achievable.

We finally note that the scaling law for the computational time varies as the fourth power of the radius of truncation, so big savings could be achieved by further reduction of the radius. For example, if we reduced the radius of truncation by a half (corresponding to truncating the basis function at a level of around 2×10^{-6}), then a further factor of 16 speed up could be achieved. By contrast, however, if we wished to increase the value of σ, in order to achieve a smoother mapping, then the computational cost would rise sharply, perhaps necessitating a more aggressive truncation threshold.

We have demonstrated in this section that the fact that the GTM M step involves a solution of linear equations does not pose a problem as far as scalability is concerned; that the matrix to invert is naturally sparse, and hence that sparse techniques can be employed to scale up the algorithm. However,

it should be emphasized that we are not claiming that GTM could be easily scaled up to the size of the network in [Kohonen et al., 2000]. As well as the matrix inversion, we need to consider the fact that the responsibility matrix needs to be calculated at each step, whereas by contrast, in the SOM, one need only search for the "winning node". It is possible that computational costs could again be reduced significantly by appealing to sparsity — the responsibility in a trained model should be localized to a "bubble" of activity. However, it is still necessary to compute all the elements of the responsibility matrix in that locality, as opposed to just searching for the winner. Kohonen et al. applied several heuristics to accelerate the search for the winning node, and a scaling up of GTM to similar sizes would necessitate the development of similar heuristics. For example, in the early stages of training, the responsibility bubble will be non-localized. However this problem could be overcome by training a succession of larger GTM models; we start with a small GTM, then scale up the size and complexity by an appropriate re-scaling procedure. This is essentially the approach adopted in the million node network of Kohonen's paper. We shall be developing a similar technique later in the chapter for scaling up a GTM through time model.

4.1.2.2 Effect of truncation on run time and conditioning

As it turns out, the truncation of basis functions beyond a certain radius not only reduces the computation time required, but also improves the conditioning of the matrix to be inverted. In the straightforward M step update of Equation (4.7), it turns out that the matrix to be inverted becomes rank deficient as the width of the basis functions increases. Table 4.1 gives results for a $M = 40 \times 40$, $K = 80 \times 80$ GTM model, with truncation of the basis functions at 3σ. The computation was run using Matlab. The GTM $\mathbf{\Phi}$ matrix was set up with various values of the smoothing parameter σ. Then $\mathbf{\Phi}'\mathbf{\Phi}$ was calculated in the first case, and $\mathbf{\Phi}'_{\text{trunc}}\mathbf{\Phi}_{\text{trunc}}$ in the second case, where $\mathbf{\Phi}_{\text{trunc}}$ is the same matrix with all values corresponding to $R > 3\sigma$ set to zero. Additionally, sparse methods were used to perform the matrix multiplication, in order to compute the speed up.

It is clear that the truncation has effectively overcome the ill-conditioning

σ	CPU (Dense)	CPU(Sparse)	RCOND(Dense)	RCOND(Sparse)
1.0	116.8	0.375	3.9151e-011	3.1541e-009
1.1	91.3	0.453	8.1024e-013	2.0624e-009
1.2	81.3	0.532	1.1543e-014	1.5039e-009
1.3	76.9	0.625	1.1567e-016	1.1669e-009
1.4	73.4	0.765	1.1101e-018	8.4868e-010
1.5	72.8	0.953	1.7565e-020	1.2567e-009
1.6	72.7	1.375	2.8529e-021	1.2308e-009
1.7	72.7	1.938	2.7482e-021	1.0020e-009
1.8	72.7	2.484	4.2845e-022	8.3466e-010
1.9	73.8	3.125	2.8516e-021	8.0771e-010
2.0	73.5	3.766	6.4915e-022	7.2898e-010
2.1	72.9	4.359	6.3249e-022	5.7041e-010
2.2	72.6	5.094	4.4262e-022	5.6719e-010
2.3	72.6	5.953	2.8329e-022	1.7025e-009
2.4	72.7	6.875	3.4046e-022	4.6419e-010
2.5	73.6	7.891	2.0067e-022	4.1503e-010

Table 4.1: Effect of truncation of the GTM basis functions on condition number. A set of equations is ill-conditioned if $1 + \texttt{RCOND} = 1$ to machine precision.

problem, with the condition number RCOND being well within the acceptable range for all values of σ. By contrast, for the dense case the matrix is rank deficient for $\sigma > 1.2$. The test for rank deficiency is if $1 + \texttt{RCOND} = 1$, to machine precision. If this is true, then the matrix is singular to working precision. The speed-up gained by using sparse matrix multiplication ranges from over 300 to around 10. It is not known why the dense evaluation took longer for the first two instances, though it is possibly due to exceptions being generated because of numerous numerical underflows.

It should be noted, however, that in practice, ill-conditioning does not present a problem, because regularization is used. This is discussed at length in [Svensén, 1998] , and corresponds to having a spherically symmetric Gaussian prior over the elements of the matrix $\mathbf{W}$:

$$p(\mathbf{W}|\lambda) = \left(\frac{\lambda}{2\pi}\right)^{MD/2} \exp\left\{-\frac{\lambda}{2}\sum_{j=1}^{M}\sum_{k=1}^{D} w_{jk}^2\right\}, \tag{4.16}$$

which leads to a modification of Equation (4.7), to incorporate the regularization term:

$$(\mathbf{\Phi}'\mathbf{G}\mathbf{\Phi} + \lambda\mathbf{I})\mathbf{W}' = \mathbf{\Phi}'\mathbf{R}\mathbf{T}. \tag{4.17}$$

In this section, we have examined ideas to do with sparse representations of the mapping produced by GTM, as applied to the static model. However, it will be seen that similar ideas can also be fruitfully applied to the temporal extension to GTM, which we discuss in the next section.

4.2 GTM Through Time

In order to achieve a temporal version of GTM, we replace the static prior of Equation (4.3) with a dynamically evolving conditional distribution $p(\mathbf{x}_{t+1}|\mathbf{x}_t)$ in exactly the same manner as for Probabilistic PCA Through Time. The difference in implementation comes about because the distribution is multinomial, defined only at the K grid points in latent space. Therefore the natural formulation of the algorithm is to use a Hidden Markov Model representation, where each of the grid points corresponds to a hidden state of the model, and state transitions are governed by a matrix of probabilities,

called the *Transition Probability Matrix*, and where the GTM mixture distribution is termed the *Emission Probability Matrix*. The model is represented schematically in Figure 4.4.

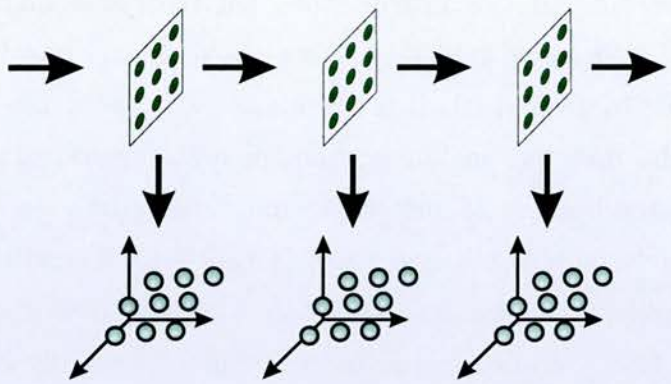


Figure 4.4: Schematic of the GTM Through Time model. GTM Through Time is a Hidden Markov Model, in which a matrix of transition probabilities governing transitions between states, is learned during the algorithm. The parameters of the GTM mixture distribution are also learned, using a similar update formula for i.i.d. GTM, but where the responsibilities take temporal context into account.

4.2.1 Related work

The original version of the GTM Through Time algorithm was published by Bishop, Hinton and Strachan in [Bishop et al., 1997a]. One of the key motivating concerns in that paper was to be able to cut down the number of adjustable parameters in the model; even for a comparatively small GTM model with a 10×10 grid of basis functions, for example, there would be $100 \times 100 = 10,000$ adjustable parameters in the HMM transition matrix. Our approach (reviewed in more detail in Section 4.2.5.1) was to define groups of states, where the probability of transition to any state in one of the groups was common to all the states in the group.

Two papers have appeared since, which are closely related to our approach, but adopt a different method for reducing the parameter count. In [Roweis, 1999], a form of *Constrained Hidden Markov Model* is adopted, where the states of the HMM are considered to be placed on a topological grid in

Latent Space. The grid (which need not be confined to 2 dimensions) is defined by a suitable sphere-packing scheme (i.e. rectangular or hexagonal for 2 dimensions, or body or face-centred cubic for 3 dimensions). Given this grid, then only restricted transitions are allowed in the model (for example to nearest neighbours, or neighbours within a certain distance in topological space). This reflects the prior knowledge that data evolves slowly through time, and so we do not expect large jumps from one part of data space to another. In Roweis's model, the transition probabilities are not learned, but are fixed at the outset, the simplest possibility being to have a uniform probability for all transitions from one state. However, it is also possible to model directed flows by discretization of the matrix linear dynamics. Roweis gives a toy example of noisy sequences of integers generated by traversal of the topological space, where each cell in the packing scheme contains an integer (and where different cells may contain the same integer). During traversal, noise is applied, and the wrong integer is occasionally output. Here, the hidden states are represented as multinomial distributions over a number of discrete symbols. A more advanced example is also given of reconstructing articulator (tongue) movements from audio speech signals, where a 4 dimensional latent space of 4096 states were used on an $8 \times 8 \times 8 \times 8$ grid. Despite the large size of the transition matrix, the constraint of having only near-neighbour transitions meant that the matrix was very sparse, and hence the problem was tractable.

A strongly related approach is given in [Kaban and Girolami, 2002], which is more related to visualization (in this application of topic evolution in internet newsgroups and IRC chat rooms), in that the latent space dimension is 2. This method also uses a fixed transition matrix, with a rectangular grid, and where the transition probability is modelled as a Gaussian function with the distance in latent space. The width parameter of the Gaussian is determined empirically. The Gaussian is truncated at a certain distance, in order, once again, to have a sparse transition matrix which is efficient computationally. Kaban and Girolami do not use a GTM formulation for the output mapping, and allow the formation of a smooth mapping to be governed by the smooth *a priori* formulation of the transition matrix. This work is reviewed in more detail in section 4.2.5.2. In particular, we shall show that the non-GTM for-

mulation of the M step of the algorithm is a limiting case of GTM through time, where the widths of the basis functions is allowed to go to zero. It was this observation that motivated the sparse representation of the GTM presented in Section 4.1.2 above.

In the following sections, we take the formulation of GTM Through Time somewhat further than in [Bishop et al., 1997a], and develop techniques that also exploit sparsity in a similar manner to the above two papers. In particular, we show that the large number of parameters at the outset is not in general a problem for GTM Through Time, as a sparse representation can be allowed to develop naturally because most of the parameters decay rapidly to zero and cannot subsequently change. If desired, we can adopt a similar approach to the above two papers by initializing the Transition matrix as a sparse matrix to allow only localized transitions, but we can also incorporate updating of the parameters, which is not a large overhead for a sparse matrix. Additionally, we shall develop a technique for scaling up a GTMTT model, by beginning with a relatively small model with a full transition matrix, allowing the sparsity pattern to develop during initial training, and then scaling up to a larger number of hidden nodes. Because we can relax the assumption that only localized transitions are allowed, we demonstrate non-local transitions being learned in a toy example in Section 4.3.

First, we review the theory behind Hidden Markov Models.

4.2.2 Hidden Markov Models

The Hidden Markov Model was discussed informally in Chapter 1, with reference to the problem of inferring whether a racehorse was on form or off form from an observed sequence of results in previous races. We now present the model more formally,broadly following [Rabiner, 1989], in the context of GTM Through Time.

A **Markov Chain** is one that is characterized by K discrete states $\{X_i; i = 1 \ldots K\}$, only one of which can be occupied at time t. The subsequent state of the system at time $t+1$ is dependent only on the state at time t. Each state X_i corresponds to one of the GTM grid points in latent space, $\mathbf{x}_i$. We shall denote the state at time t by the symbol q_t.

In a **Hidden Markov Model**, the states X_i are not directly observable, but can only be inferred from a time sequence of the observations: $\mathbf{Y} = \{\mathbf{y}_1\mathbf{y}_2 \ldots \mathbf{y}_T\}$. Thus the sequence of states $\mathbf{X}$ corresponds to a path through latent space, passing at each point through one of the discrete grid points. One method of using GTM Through Time to visualize the sequence of observations $\mathbf{Y}$ will be to compute the most likely path through states and plot this discretized path. An alternative visualization can be achieved by computing the posterior probability distribution of the states at each time, and then plotting the mean.

We define the probability of transition from state i to state j as A_{ij}, and hence the probabilities for all possible transitions are defined as a matrix $\mathbf{A}$, which is called the **transition matrix**. The transition matrix gives a prediction of the next state of the system, given the previously inferred state, in the absence of the next observation. This prediction is analogous to the time-update step in the Kalman filter based linear dynamical system model of Chapter 3.

We also need to define the probability of the observation, given a particular state. In the case of discretized observations (like the outcome of a horse race), then this may be represented as a matrix, which is formally given the symbol $\mathbf{B}$. However, in GTM Through Time, we are dealing with the case of continuous observations, and $\mathbf{B}(\mathbf{y}_t)$ is be a vector of probabilities that is the output of the standard GTM probability density model. The probabilities associated with $\mathbf{B}$ are termed the **emission probabilities**, and they are directly analogous to the measurement update in the Kalman filter formulation.

To complete the definition of the Hidden Markov Model, we need to have the prior state probabilities, in order to start the inference chain. These are conventionally denoted with the symbol $\boldsymbol{\pi}$. Note that in standard GTM, the prior over the state variables (i.e. the mixing coefficients of the Gaussian Mixture Model), is taken to be uniform over the K components. In the dynamical case, we are able to learn this distribution as part of the EM algorithm.

Hence the HMM is fully defined by a parameter set $\lambda = \{\mathbf{A}, \mathbf{B}, \boldsymbol{\pi}\}$. It may be represented as a probabilistic graphical model shown in Figure 4.5.

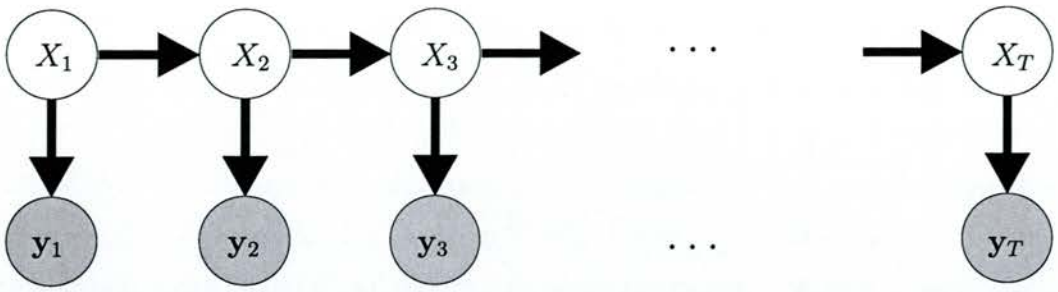


Figure 4.5: HMM represented as a Probabilistic Graphical model. The state variables X_i are numbers in the range $\{1 \ldots K\}$, corresponding to each of the grid points $\mathbf{x}_i$ in latent space. The observation vectors are the $\mathbf{y}_j$ variables. The horizontal arrows correspond to predictive inferences based on the previous state, and the transition probabilities $\mathbf{A}$. The vertical arrows correspond to measurement updates, where the inferred values of the latent variables are updated according to the emission probabilities $\mathbf{B}$. The first state is initialised with the prior distribution $\boldsymbol{\pi}$.

An important property of the HMM learning algorithm is that a zero entry in the transition matrix $\mathbf{A}$ always stays at zero during the re-estimation. Hence it is possible to encode some prior knowledge of the type of transitions that are allowable, by specifying the structure of the transition matrix at the outset. For example, in a speech-recognition HMM, a model that recognises a sequence of phonemes, all of which must be present, might have a bi-diagonal transition matrix specified at the outset, corresponding only to self-transitions (in the middle of a single phoneme), and transitions to the next state in the sequence (i.e. to the next phoneme). The absence of elements in further diagonals above this indicates the knowledge that phonemes are not skipped by the speaker, and the absence of entries in the lower triangle of the matrix indicates that earlier phonemes are not repeated. Similarly, we could encode our prior knowledge of the slow evolution of time-dependent data by specifying a locality constraint, so that:

$$A_{ij} = P(q_t = X_i, q_{t+1} = X_j) > 0 \quad \text{if} \quad \|\mathbf{x}_i - \mathbf{x}_j\| < R, \tag{4.18}$$

in other words, only transitions that fall within a radius R in latent space are allowed by the model. This has the added advantage drastically reducing

the computation time for the HMM training algorithm, as computations need only be performed for the non-zero entries.

4.2.3 The learning algorithm for the HMM

The parameters of a Hidden Markov Model may be learnt through an EM algorithm, where at each M step the parameters of the model are maximized, given the current expectation values of the latent variables. The expectation values are computed in the E step through an algorithm that propagates forwards and backwards through time.

4.2.3.1 Computation of likelihood

Forward propagation through time allows computation of the likelihood of the data given the model $P(\mathbf{Y}|\lambda)$, and is also used when the HMMs are used to perform classification tasks. It computes along the way a set of variables that are used in the re-estimation procedure.

We define the *forward probabilities* α_t^i as:

$$\alpha_t^i = P(\mathbf{y}_1\mathbf{y}_2 \ldots \mathbf{y}_t, q_t = X_i|\lambda) \tag{4.19}$$

This is the probability of the partial sequence of observations from time 1 up to t, *and* the state being X_i at time t. These probabilities can be calculated inductively using the emission and transition matrices $\mathbf{B}$ and $\mathbf{A}$ by the following recursion:

$$\boldsymbol{\alpha}_1 = \boldsymbol{\pi} \odot \mathbf{B}(\mathbf{y}_1) \tag{4.20}$$

$$\boldsymbol{\alpha}_{t+1} = (\mathbf{A}'\boldsymbol{\alpha}_t) \odot \mathbf{B}(\mathbf{y}_{t+1}), \tag{4.21}$$

where $\odot$ denotes elementwise vector multiplicaton. In (4.20), the first state distribution is estimated using the prior. Equation (4.21) then propagates through time, the bracketed term being directly analogous to the time update equations in the Kalman Filter equations (3.41) and (3.42), except that here the whole distribution is updated in a single step by multiplication by the transition matrix. The post multiplication by the vector of emission probabilities is analogous to the Kalman Filter measurement update equations (3.44) and (3.45).

Computation of the log likelihood can in principle can be achieved simply by summing the probabilities contained in $\boldsymbol{\alpha}_T$:

$$P(\mathbf{Y}|\lambda) = \sum_{k=1}^{K} \alpha_T^k \tag{4.22}$$

However, the implementation of this recursion requires care, because of the limited dynamical range of precision in floating point arithmetic. Therefore, in practical implementations, it becomes necessary to introduce rescaling of the α_t periodically, in order to avoid numerical underflow. Accordingly, scale factors

$$c_t^\alpha = \sum_{k=1}^{K} \alpha_t^k, \tag{4.23}$$

are introduced at each time step, to renormalize the forward probability vectors to sum to unity. As a convenient side-effect, this allows the log likelihood to be calculated during the recursion, but summing the logarithms of the scale factors at each step.

4.2.3.2 Re-estimation of the model parameters

In the EM algorithm for training GTMTT, we need to compute the expected values of the latent variables, which are formulated as a set of responsibilities, with temporal context taken into account. These are defined as $\gamma_t^i = P(q_t = X_i|\mathbf{Y}, \lambda)$, i.e. the probability that the state is X_i at time t, given the model and the entire observation sequence.

In order compute these responsibilities *backward variables* β_t^i, are introduced:

$$\beta_t^i = P(\mathbf{y}_{t+1}\mathbf{y}_{t+2}\cdots\mathbf{y}_T|q_t = X_i, \lambda) \tag{4.24}$$

In other words, this is the probability of the partial observation sequence from time $t+1$ to T, given the state X_i at time t. We can then compute, the probability of being in state X_i at time t, given the entire observations sequence by combining the forward and backward probabilities thus:

$$\gamma_t^i = P(q_t = X_i|\mathbf{Y}, \lambda) = \frac{\alpha_t^i \beta_t^i}{P(\mathbf{Y}|\lambda)} \tag{4.25}$$

These quantities will be the responsibilities used in the M step of the algorithm, and will be substituted for $\mathbf{R}$ in Equation (4.7).

The computation of the backward variables is directly analogous to the backward calculation of the state variable estimates in the Kalman Smoothing algorithm of the previous chapter, via the Rauch recursions, as the calculation takes into account past and future temporal context.

The $\boldsymbol{\beta}$ vectors are computed via a similar inductive algorithm to the forward variables:

$$\beta_T^i = 1, \qquad 1 \leq i \leq K \tag{4.26}$$

$$\boldsymbol{\beta}_t = \mathbf{A}(\mathbf{B}(\mathbf{y}_{t+1}) \odot \boldsymbol{\beta}_{t+1}) \tag{4.27}$$

The variables are initialized to unity because, given the state at the last time step, there are no further observations. The bracketed term in (4.27) is the observation update from the previously predicted vector of probabilities, and the pre-multiplication by the transition matrix $\mathbf{A}$ produces a new predicted vector of state probabilities for the previous time step, given the rest of the sequence.

Like the forwards recursion, the backwards recursion also requires a rescaling procedure to prevent numerical underflow. In standard versions of the algorithm, the recommended procedure is to re-use the scaling parameters for the $\boldsymbol{\alpha}$ vectors computed in the forward pass, to rescale the $\boldsymbol{\beta}$ vectors, [Rabiner, 1989]. This procedure means that the scaling factors are all cancelled out in the re-estimation formulae, which are unaffected.

However, for GTM Through Time this simple procedure has proved inadequate, as it can lead to numerical overflow during the back propagation recursions. This is because GTM is a highly constrained model with many state variables. It can happen during the training, when using real data, that certain outlier data points can result in very low values of the emission probabilities for all the states. This would not normally be a problem with a standard mixture component HMM, because the mixture component variances are all updated independently, and so some can be large enough to accommodate outlier points. In the case of the standard GTM, all the variances are constrained to be the same, and so some points can have extremely low probability for all the mixture components (i.e. are placed a long way from the GTM manifold). In the standard GTM, this is not a problem; the observation vector in question has negligible effect on the re-estimation

procedure, because the responsibilities are smeared out over the whole set of states. However, it causes a problem in the HMM equations, because the corresponding $\boldsymbol{\alpha}_t$ variables all have a very low value. Suppose that observation $\mathbf{y}_t$ is an outlier that causes very low values of the elements of $\boldsymbol{\alpha}_t$. Then the corresponding value of the scale factor c_t^α from (4.23) is very small. However, when we come to rescale $\boldsymbol{\beta}_t$ from (4.27), the computed value does not depend on the outlier observation $\mathbf{y}_t$ but only on the values from $t+1$ onwards. Hence the unscaled value of $\boldsymbol{\beta}_t$ is not correspondingly small, and when dividing c_t^α, an overflow can result.

Since removal of the outlier point would also necessitate the removal of the whole observation sequence, which otherwise contains useful training data, we adopt a slightly different approach, which is to use a separate set of scale factors for the backwards recursions, to rescale the backward variables at each step to sum to unity:

$$c_t^\beta = \sum_{k=1}^{K} \beta_t^k. \tag{4.28}$$

Finally, a re-estimation procedure for the transition matrix probabilities A_{ij} is required. Introducing the term $\xi_t^{i,j}$, which is the probability of being in state X_i at time t and state X_j at time $t+1$, given the model and the observation sequence, we have:

$$\xi_t^{i,j} = P(q_t = X_i, q_{t+1} = X_j | \mathbf{Y}, \lambda) \tag{4.29}$$

This can be calculated in terms of the forward and backward variables, with reference to Figure 4.6. In order to calculate $\xi_t^{i,j}$, we must multiply together the probabilities of four events, namely:

1. Being in state X_i at time t given the partial observation sequence till then (via the forward probabilities).

2. The transition $X_i \rightarrow X_j$ (via the transition probabilities)

3. The probability of being in state X_j, given the observation $\mathbf{y}_{t+1}$ (via the emission probabilities).

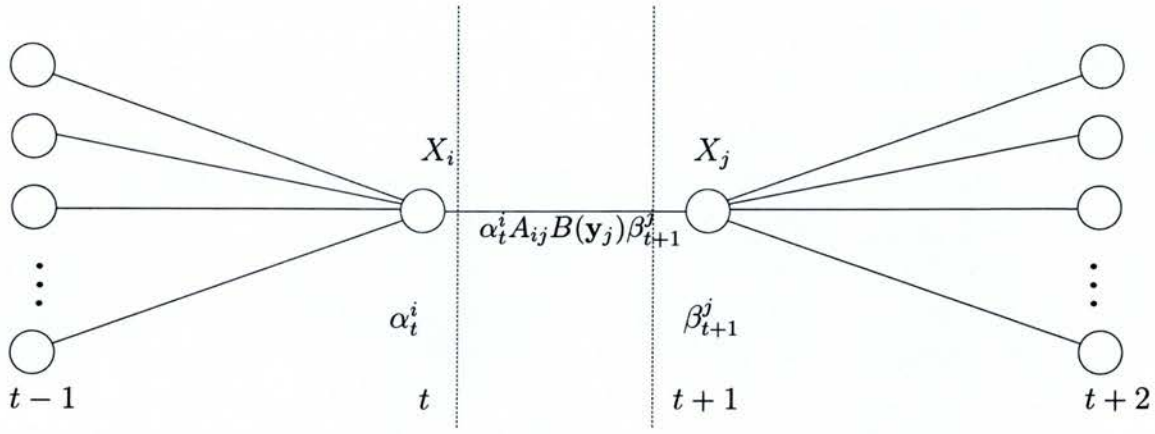


Figure 4.6: Trellis diagram illustrating the probability terms to multiply together to compute the joint probability of being in state X_i at time t and X_j at time $t+1$, given the observation sequence and the model.

4. The probability of the partial sequence from $t+1$ to the end given state X_j (via the backward probabilities).

Hence by examination of Figure 4.6, $\xi_t^{i,j}$ can be written:

$$\xi_t^{i,j} = \frac{\alpha_t^i A_{ij} B_j(\mathbf{y}_{t+1}) \beta_{t+1}^j}{P(\mathbf{Y}|\lambda)}, \tag{4.30}$$

or, equivalently, in matrix-vector notation as:

$$\boldsymbol{\xi}_t = \frac{\mathbf{A} \odot \left[\boldsymbol{\alpha}_t (\mathbf{B}(\mathbf{y}_{t+1}) \odot \boldsymbol{\beta}_{t+1})'\right]}{P(\mathbf{Y}|\lambda)}, \tag{4.31}$$

where the symbol $\odot$ here represents element-wise matrix multiplication.

From the definitions of these variables, it is clear that by summing over time we can compute expected numbers of transitions with respect to the states. Thus $\sum_{t=1}^{T-1} \gamma_t^i$ is the expected number of transitions from state X_i during the time sequence, and $\sum_{t=1}^{T-1} \xi_t^{i,j}$ is the expected number of transitions from state X_i to state X_j.

This leads to simple re-estimation formulae for the transition matrix; A_{ij} is computed by dividing the expected number of transitions $X_i \rightarrow X_j$ by the expected number of transitions from X_i:

$$A_{ij}^{\text{new}} = \frac{\sum_{t=1}^{T-1} \xi_t^{i,j}}{\sum_{t=1}^{T-1} \gamma_t^i} \tag{4.32}$$

Note that the scaling factions c_t^{α} and c_t^{β} are implicit in the formulae given here, and in the implementation have to be incorporated with some care to prevent overflow in this calculation.

Likewise, the prior state probabilities $\boldsymbol{\pi}$ can be re-estimated by computing the expected number of times in X_i at time $(t = 1)$:

$$\pi_i = \gamma_1^i \tag{4.33}$$

The re-estimation formulae (4.32) and (4.33) can be shown to maximize $P(O|\lambda)$ by treating the problem as a constrained optimization problem, subject to the constraints that the priors, and the rows of the transition matrix must sum to unity.

The equations given only relate to a single observation sequence, but the generalization to multiple observation sequences is very straightforward. Since the formulae are based on frequencies of occurrence of state transition events, we simply add together the numbers of occurrences for each individual observation sequence in the re-estimation formula.

4.2.3.3 Summary of the GTM Through Time Basic Algorithm

We now have all the components required for the basic EM algorithm for GTM Through Time. To summarise:

The E-step uses the HMM forward/backward recursions. First, we compute the static GTM responsibilities from the standard GTM E-step (4.9). These are then used as the emission probabilities in the E step in the HMM. The forward recursions use equations (4.20) and (4.21), with rescaling of $\boldsymbol{\alpha}_t$ according to Equation (4.23). The backward recursions use equations (4.26) and (4.27), with rescaling according to Equation (4.28). On the backward pass, the posterior probabilities $\boldsymbol{\gamma}_t$ are computed according to Equation (4.25), and the matrices $\boldsymbol{\xi}_t$ are computed from Equation (4.31), taking into account the scaling factors.

In the M-step, the GTM parameters are re-estimated using the standard GTM re-estimation formulae (4.7), and (4.8), with the difference that the i.i.d. GTM responsibilities are replaced by the γ_t computed from the HMM.

Finally, the transition matrix $\mathbf{A}$ is re-estimated from (4.32), and the prior state probability vector $\boldsymbol{\pi}$ is re-estimated from (4.33).

The process is repeated until convergence.

4.2.4 Discussion of Basic Algorithm

4.2.4.1 Computational complexity

Firstly, as presented here, it is a much more computationally demanding algorithm than the standard GTM, which is in turn more computationally demanding than the SOM (though as demonstrated earlier, it has the same scaling properties as the SOM if sparse representations are used).

In the GTM through time algorithm, the majority of the computing cost is in the time propagation equations of the Hidden Markov Model. If there are N observation vectors in the training set (which may be broken up into a number of separate observation sequences), and K states in the model, then the forward and back propagation equations will each require $\mathcal{O}(K^2N)$ operations to compute, as will the calculation of the $\boldsymbol{\xi}_t$ matrices used to re-estimate the transition matrix. It should also be noted that K is the number of grid points on a two dimensional grid, and in increasing the model order, one increases the number of grid divisions on each axis, and the computational complexity of the model will therefore grow as the fourth power of the grid linear dimension. Some of these difficulties are overcome by the suggestions of the next section.

4.2.4.2 Storage requirements

The storage requirements for GTM through time are also apparently very demanding. A straightforward implementation of the algorithm will require storage for the matrices $\mathbf{B}$, $\boldsymbol{\alpha}_t$, $\boldsymbol{\beta}_t$, $\boldsymbol{\gamma}_t$, and $\boldsymbol{\xi}_t$. The first four of these require $\mathcal{O}(KN)$ storage locations, and the last requires $\mathcal{O}(K^2N)$. By contrast, the standard GTM algorithm only requires storage of a single matrix of size KN storage locations, for the responsibilities, which can be overwritten by the new distance calculations in the update equations for the inverse variance.

However, most of the storage problems can be overcome by efficient re-use of memory. We note that the $\boldsymbol{\beta}_t$ matrices are only required once at each time

step on the backward propagation equations, and that in the backward pass, the $\boldsymbol{\alpha}_t$ matrices can be overwritten with the newly estimated $\boldsymbol{\gamma}_t$ matrices. Furthermore, there is no need to store each individual time step value for the $\boldsymbol{\xi}_t$ matrices, so they can be summed into a single matrix at each time step. However, the emission probabilities are required on both the forward and backward pass, so the storage required is at least $\mathcal{O}(2KN)$. In the simulations carried out in this study, it was noted that the computation of the emission probabilities was not a significant amount of the overall time, and in order to achieve large models, a trade-off was performed of CPU time against storage, by recomputing the emission probabilities "on the fly" on both the forward and backward passes, and also to compute the distance matrix on the fly on the update step for the inverse variance (4.8). For models where the sheer storage requirements would have exceeded the core storage of the computer (500MB), this proved a faster option than computing the quantities only once and using the hard disk as out-of-core storage.

4.2.4.3 Parameter count

The most serious objection that might be raised against the proposed algorithm is the high number of parameters that have to be re-estimated. Even for a moderately coarse GTM grid of 10×10 points, the HMM transition matrix will be 100×100, meaning that there would be 10,000 parameters to learn. This is vastly more than for the straightforward GTM, which, for a 10×10 grid, might have 5×5 non-linear basis functions, plus two linear and one constant basis function. For a data dimensionality of $D = 5$, this would result in just 140 parameters for the weight matrix, plus the inverse variance parameter.

It might be thought that this would lead to a serious problem of overfitting the data, but it turns out not to be the case, because in practice it has been found that most of the transition matrix parameters decay rapidly to negligible values. To see why this is the case, we have to consider the nature of the re-estimation formula for the transition matrix entries. The value A_{ij} is re-estimated according to the formula:

$$A_{ij}^{\text{new}} = \frac{\text{Expected number of transitions from } X_i \rightarrow X_j}{\text{Expected number of transitions from } X_i} \tag{4.34}$$

Hence the estimate is based on event counting, and the numerator is based on the product of individual terms shown in Figure 4.6:

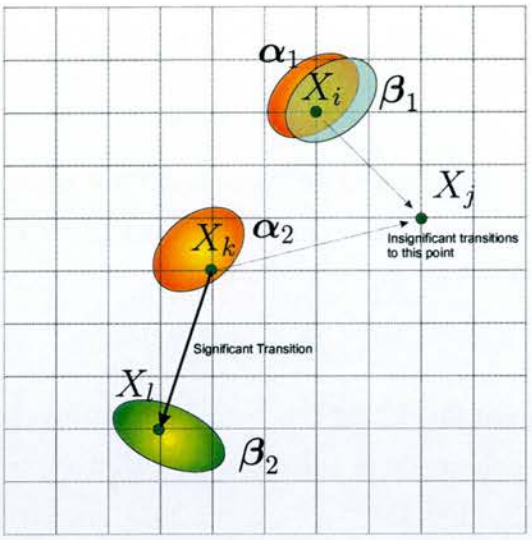
$$\xi_t^{i,j} = \alpha_t^i A_{ij} \mathbf{B}(\mathbf{y}_j) \beta_{t+1}^j \tag{4.35}$$

A key motivation for taking temporal context into account is the assumption of temporal coherence of the data, and that successive data vectors are highly correlated. Therefore, we expect that transitions from a particular state X_i will only take place to a very small number of other states.

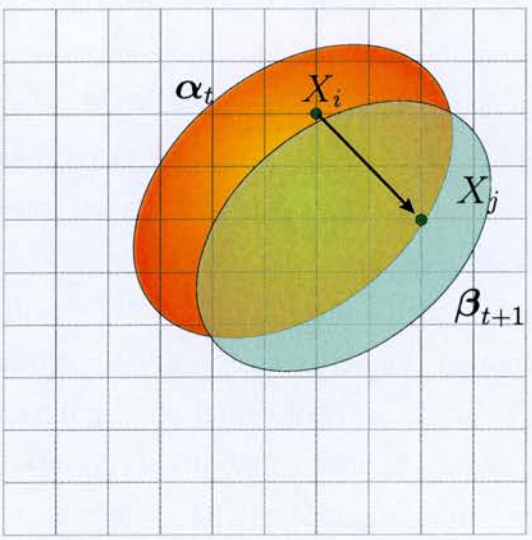
In most cases, we also expect the data to vary smoothly, and hence for transitions from state X_i, we should only expect contributions to $\xi_t^{i,j}$ and $X_j \in \mathcal{N}(X_i)$, where $\mathcal{N}(X_i)$ denotes those grid points in the local neighbourhood of state X_i. This is illustrated schematically in the top right hand corner of Figure 4.7(a), where the orange ellipse $\boldsymbol{\alpha}_1$ depicts the forward probability distribution over the GTM grid at a given time, and the overlapping green ellipse represents the backward probability distribution $\boldsymbol{\beta}_1$ at the next time step . Only the small submatrix $\xi_t^{\mathcal{N}(X_i),\mathcal{N}(X_i)}$ will receive contributions (here we have dropped the suffix t to indicate summing all contributions). The isolated grid point X_j will not contribute significantly to $\xi^{i,j}$ because X_j lives in the extreme tails of the probability distributions.

Finally we note from Equation (4.34), that the re-estimated transition matrix elements for isolated cases like X_j will also be very small, and this will reinforce the effect the next time round as the newly estimated element A_{ij} will be used again in the frequency counting of Equation (4.35). The outcome will be to drive A_{ij} rapidly to a negligible value over a sustained period of iterations. This is precisely what we observe in practice.

The case where successive data points undergo a sudden transition, though much rarer in practice, is illustrated by the disjoint ellipses in Figure 4.7(a), labelled $\boldsymbol{\alpha}_2$ and $\boldsymbol{\beta}_2$. Again, here we expect the small submatrix $\xi_t^{\mathcal{N}(X_k),\mathcal{N}(X_l)}$ to receive contributions, and the points in the tails of the probability distribution, such as X_j will again not contribute significantly.



(a) Localized probability density



(b) Widely spread probability density

Figure 4.7: Illustration of the frequency counting required for re-estimation of the transition matrix parameters. Diagram (a) shows the case where "activity bubbles" of the forward and backward probabilities are highly localised, for the cases where data evolves smoothly, and via a discontinuous jump, resulting in the re-estimate for most elements in the matrix driven rapidly to zero. Diagram (b) illustrates the corresponding case where the activity bubbles are spread out due to a "stiff" GTM mapping. In this case the smoothly varying nature of the distributions involved results in the re-estimated parameters to obey a smoothness constraint.

However, it will be the case that more transition matrix parameters will have significant values if the GTM model is constrained so that it is very inflexible, by having a small number of non-linear basis functions and a large basis function width. This corresponds to the case where we have the prior assumption that there is a large amount of noise in the data, which we do not wish the GTM manifold to attempt to fit. Thus we only allow very moderate curvature. Correspondingly, the variance of the GTM noise model will be large, and we shall expect the emission densities, and hence the forward and backward probability distributions, to be smeared out over a large area. This is illustrated in Figure 4.7(b).

In this situation, we do expect that the submatrix $\xi_t^{\mathcal{N}(X_i),\mathcal{N}(X_j)}$ will contain a large number of elements because the neighbourhoods of X_i and X_j for the forward and backward distributions will encompass a large number of grid points. However, we also note that the entries in this matrix will vary smoothly over the grid, as the emission densities generated by the stiff GTM also vary smoothly.

Hence, even in this case, where there are a large number of adjustable parameters in the transition matrix, we do not expect there to be overfitting, because they are constrained to vary smoothly over the grid, by the stiff nature of the GTM mapping. Viewed from a Bayesian perspective, this is equivalent to stating that the choice of parameters in the GTM mapping imposes a prior distribution on the transition matrix weights that causes them to vary smoothly. In principle, one could then compute the number of *well determined parameters*, [Bishop, 1995, p 410], whose values are determined by the data, rather than by the prior. There is an extensive discussion of parameter selection from a Bayesian viewpoint in [Svensén, 1998, p 72ff], where the framework of [MacKay, 1992] is adapted in order to estimate the inverse variance parameter, and the regularization parameter for GTM. However, no simple solution could be found for the basis function widths, which were determined empirically via a grid-based search. In the light of this conclusion it does not seem likely that the framework could be adapted further to consider the prior over transition matrix values.

4.2.5 Alternatives to the basic algorithm

In the previous section it was observed that the algorithm is computationally demanding, both in CPU time required, and also in storage requirements. Although the high parameter count has been shown not to be a cause of over-fitting, there is still motivation for finding ways of reducing the complexity. These are now considered.

4.2.5.1 Reducing the parameter count by grouping

In [Bishop et al., 1997a], an alternative approach, allowing the parameter count to be reduced directly, and thereby incorporate prior knowledge, was introduced. This was achieved by allowing groups of transitions from a particular state to be governed by a common parameter. We denote the kth group from state X_i as $\mathcal{G}_{ik}$, and the common probability for transitions $(X_i \rightarrow X_{j \in \mathcal{G}_{ik}})$ to be η_{ik}, satisfying $\sum_k \eta_{ik} = 1$. If we introduce indicator variables C_{ikj} equal to 1 if $X_j \in \mathcal{G}_{ik}$, and 0 otherwise, then the transition probability of $(X_i \rightarrow X_j)$ is given by:

$$A_{ij} = \sum_k \eta_{ik} C_{ikj} N_{ik}^{-1}, \tag{4.36}$$

where N_{ik} denotes the number of states in group $\mathcal{G}_{ik}$. This leads to a simple modification of the transition probability parameters, to re-estimate the grouped transition parameters:

$$\eta_{ik} = \frac{\sum_t \sum_{j \in \mathcal{G}_k} \xi_t^{i,j}}{\sum_k \sum_t \sum_{j \in \mathcal{G}_k} \xi_t^{i,j}}. \tag{4.37}$$

This algorithm allows prior knowledge to be incorporated in the choice of the groupings. For example, we might choose to allow localized transitions, resulting in groups of one state in the neighbourhood of X_i, and the occasional transition to a more remote region, governed, for example, by a single group.

Alternatively, we could impose a more symmetric constraint, whereby each group from X_i could be a $n \times n$ block of states, where n divides evenly into $\sqrt{K}$. In fact this approach reduces the computational requirements by allowing the forward and backward equations to be computed by a $K \times N_{\text{groups}}$ matrix. This can be done only if the members of the groups $\mathcal{G}_{ik}$

are independent of i, when the backward probability matrices can be defined as:

$$\beta_t^{\mathcal{G}_k} = P(\mathbf{y}_{t+1}\mathbf{y}_{t+2}\ldots\mathbf{y}_T|q_t \in \mathcal{G}_k, \lambda) \tag{4.38}$$

In the light of the preceding analysis, it transpires that the motivation for reducing the parameter count is solely to reduce the computational load, and so this approach only finds applicability either where some explicit prior knowledge can be encoded into the grouping structure, or where a speed-up is desired at the expense of adding a blocked structure to the transition probabilities.

4.2.5.2 Fixed transition matrix and standard Gaussian Mixtures update

This approach was suggested in [Kaban and Girolami, 2002] as a simpler alternative to GTM through time. Kaban and Girolami replace the GTM re-estimation step with a standard Gaussian Mixtures update rule, and keep the transition matrix fixed, utilizing the prior knowledge of smooth variation of data through time to determine the transition probabilities at the outset. Denoting the latent space locations of the states X_i as the vectors $\mathbf{x}_i$, they set up the probabilities in the transition matrix at the outset as a Gaussian function of the distance between states in latent space:

$$A_{ij} \propto \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x}_i - \mathbf{x}_j\|^2\right), \tag{4.39}$$

where the width parameter σ defines the size of the neighbourhood where local transitions occur.

The key idea behind the approach is that the topographic ordering is produced solely by the pre-defined transition probabilities, rather than by the GTM algorithm.

The algorithm also offers the computational advantage that if desired, the Gaussian functions governing the transition parameters can be truncated at a suitable radius, giving a sparse representation for the transition matrix. If the neighbourhood is sufficiently small compared to the overall size of the matrix, then this will offer considerable speed-ups in the forward and

backward propagation recursions through time, which are a major overhead in the computation.

The correct value of the width parameter σ has to be determined empirically, as there is no analytical formula for re-estimation via the M step.

This approach can be understood as a limiting form of the GTM through time algorithm, where the number of states K is set equal to the number of basis functions M, and the basis function width is allowed to tend to zero, and the re-estimation step is omitted for the transition probabilities.

While noting that the parameter count for the transition matrix is now down to 1 (the width parameter), there is a compromise involved in producing a symmetrical transition matrix. Such a system would not be capable of distinguishing between a time sequence of vectors and the same sequence in reversed time. This could be overcome by allowing re-estimation of the transition probabilities at each iteration, while initialising with the truncated Gaussians of Equation (4.39).

This idea motivates the approach given in the next section.

4.2.5.3 Sparse representations

As we have noted in the previous sections, the re-estimation formulae for the transition probabilities rapidly drive the probabilities of the irrelevant transitions to negligible values. In practice, in a machine with finite rounding error, these values become numerically rounded to zero, and hence, in a similar manner to the discussion in Section 4.1.2, we find that the transition matrix acquires an increasingly sparser structure as the training proceeds. Furthermore, it is clear from Equation (4.31), that as the formula is multiplied by the old transition probability, that if this is zero, all further estimates stay at zero.

Thus if sparse matrix techniques are used to perform the forward and backward recursions, the computational load may be reduced considerably.

So here, in a similar manner to [Kaban and Girolami, 2002], we may initialize the transition matrix to have only localized transitions with non-zero probability. But in contrast to Kaban and Girolami, we initialize the matrix with random values, and allow re-estimation to be done according to

the standard formula. This does not impose a great overhead, because in Equation (4.31) we need only compute the values of $\xi_t^{i,j}$ that correspond to the non zeros of the transition matrix.

As noted above, the transition matrix will "naturally" become sparse during training because of the finite precision of floating point arithmetic. However, the fact that the computation time scales linearly with the number of non-zeros in the transition matrix motivates the search for a more appropriate pruning strategy. It would be a waste of resources to spend most of the time multiplying numbers together of $O(10^{-100})$.

For the simulations in this thesis, we have adopted a more aggressive thresholding strategy, based on a different numerical parameter ϵ, which is the smallest number such that:

$$1 + \epsilon > 1 \quad \text{to machine precision.} \tag{4.40}$$

Accordingly, we set a pruning threshold of ϵ/K, and set values less than this to zero. The motivation for this is that the elements of each row of the $K \times K$ matrix must sum to unity, as they are probabilities. In most cases, only a very few elements of each row have significant values, leaving most of them with insignificant values. If there are approximately K of these, then their sum will have no effect on the total, because the inequality of Equation (4.40) will not be satisfied.

In reality, of course, a weighted sum of the probabilities is used in the HMM equations, so this analysis is only approximate. However, a comparison was run to check out the difference between the two methods. A 10,000 point data set (the two-link robot manipulator example from below) was generated, and used to train a GTM through time model with $M = 49$ and $K = 256$, with a basis function width of $\sigma = 2.5$ grid separations. Training was continued for 25 iterations, at which point the likelihood curve had levelled out. Two runs were performed, one with the pruning strategy applied, and another with no pruning. The curve of log likelihood looked identical for each run, and detailed comparison showed that for most of the run, the relative difference was $O(10^{-8})$, though in the last few iterations, this rose to $O(10^{-6})$. However, there was a marked difference in the achieved

	Non-zeros	Sparsity Fraction	Run Time
Prune $A_{ij} <$ realmin	19240	29.36%	618 secs
Prune $A_{ij} < \epsilon/K$	4686	7.15%	407 secs

Table 4.2: Comparison of pruning strategies

sparsity of the transition matrix, and hence the run-time. The comparison between the two is shown in Table 4.2.

4.2.6 Hierarchical model generation

This approach is an alternative to initializing with a sparse transition matrix, and may be adopted if we do not have the prior knowledge of the kind of transitions that will occur. This could happen if occasional large transitions are possible in the data set. It is not straightforward to predict where in latent space such transitions will occur, as that will depend on the shape of GTM mapping that has been learned. The approach of Section 4.2.5.1 allows occasional large jumps, but either with a fixed probability of a non-local transition to anywhere in latent space (or with some arbitrary carving up of latent space into regions). In either case, it is difficult to encode prior knowledge.

In the hierarchical approach, we initially train a small GTM through time model (for example with a 10×10 grid in latent space), and use the learned model to form the initial estimate for a larger model. In this way, the "prior knowledge" of the larger model has been inferred in the training of the smaller model, which we initialize with a dense transition matrix, and apply a pruning strategy. In order to achieve this, we need to be able to initialize the larger model from the smaller model.

For the GTM parameters, this is very straightforward. If we wish solely to increase the number of states in the model, then the $\mathbf{\Phi}$ matrix is re-computed using the new grid of states. The weights matrix of the RBF mapping is unchanged, and the new GTM manifold will simply have new mixture centres that are interpolated between the old ones.

If, however, we wish to add further basis functions to the GTM mapping, to allow a more complex mapping to be derived, then the procedure is sim-

ilarly straightforward; we compute the locations of the mixture components $\mathbf{Y}_{\text{new}}$ from the RBF mapping of the old model, based on the new grid locations. We then compute the $\mathbf{\Phi}_{\text{new}}$ matrix for the new model and then solve the Equation $\mathbf{\Phi}_{\text{new}}\mathbf{W}_{\text{new}} = \mathbf{Y}_{\text{new}}$ by standard linear techniques. So the new model is initialized to produce the same mapping as the old model, but with a different number of basis functions.

The initialization of the new HMM parameters $\{\mathbf{A}, \boldsymbol{\pi}\}$ is also straightforward, bearing in mind that we only wish to have a reasonable estimate for the starting point, based on the knowledge acquired in the training of the small model. Many schemes could be adopted for interpolating the probabilities onto the new latent space grid. Each row of the transition matrix is a probability distribution for the predicted next state, given the current state and no measurement. We could treat this as a discretized approximation to a continuous distribution, and interpolate smoothly. Alternatively, a simple procedure can be adopted if we have a rectangular grid of points, and we double the linear dimension of the grid. This is illustrated in Figure 4.8.

Essentially, each state X_i in the old grid becomes four new states $S'_{\mathcal{P}(i)}$ in the new grid, where $\mathcal{P}(i)$ denotes the set of state numbers in the larger grid corresponding to node i in the smaller grid , and so for each transition $X_i \rightarrow X_j$, we have a new set of 16 transitions $X_{\mathcal{P}(i)} - X_{\mathcal{P}(j)}$, where:

$$A^{\text{new}}_{k\in\mathcal{P}(i),l\in\mathcal{P}(j)} = \frac{A^{\text{old}}_{ij}}{4} \tag{4.41}$$

4.3 Examples

In this section we present the results of applying the GTM through time algorithm to model two synthetically generated data sets, namely:

- The **Lorenz Strange Attractor**; a chaotic non-linear dynamical system, which varies continuously in time, but at differing rates of change,

- The **Switching State Robot**; an artificial example of a robotic arm that undergoes both continuous and discontinuous changes.

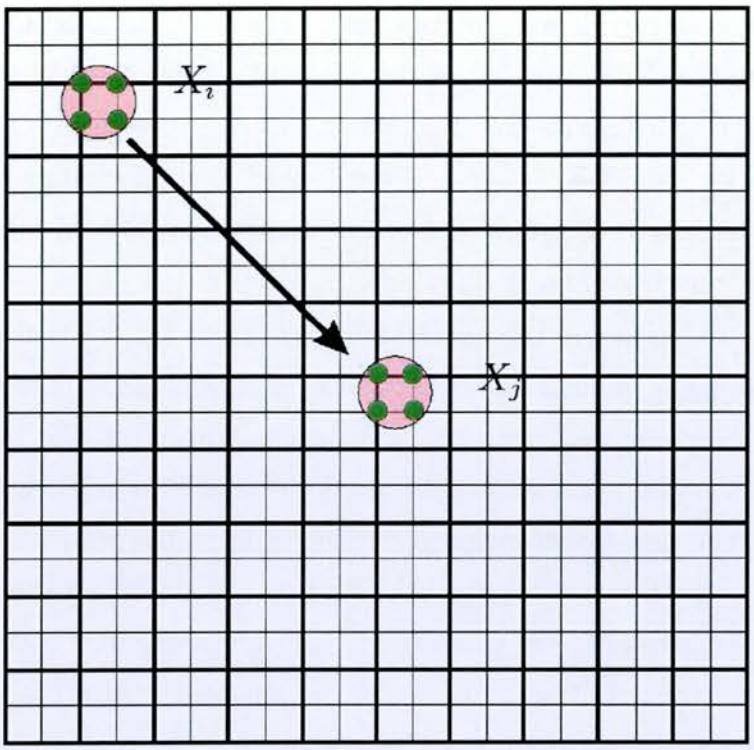


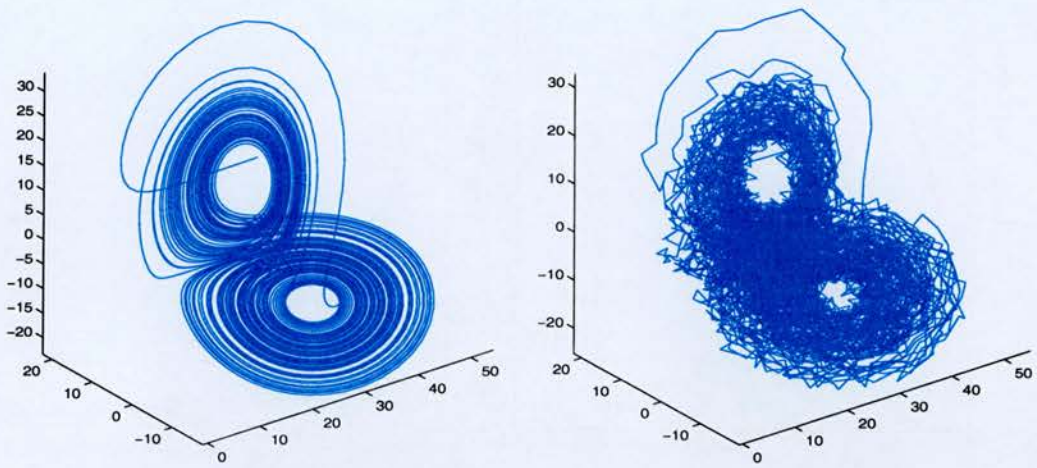
Figure 4.8: A simple scale up procedure for a transition matrix for a 20×20 grid based on a previously learned transition matrix for a 10×10 grid.

4.3.1 Lorenz Strange Attractor

The first example we present is of the well-known Lorenz Strange Attractor, which is a simple model of atmospheric convection. This was generated by the solution of the characteristic set of non-linear coupled differential equations, and adding a large amount of measurement noise. Details are given in Section A.1.

The raw data is shown, with and without added noise, in Figure 4.9.

For this illustration, a GTM through time model was initialized with a 7×7 grid of non-linear basis functions, and a 20×20 grid of latent states. The model was initialized with standard principal components techniques used for the static GTM algorithm, where the weights matrix $\mathbf{W}$ is initialized so that the initial positions of the mixture components span the 2-d principal components space of the data. The transition matrix was initialized as a dense 400×400 matrix, and the model was trained for 25 iterations of the EM



(a) Clean (b) Noisy

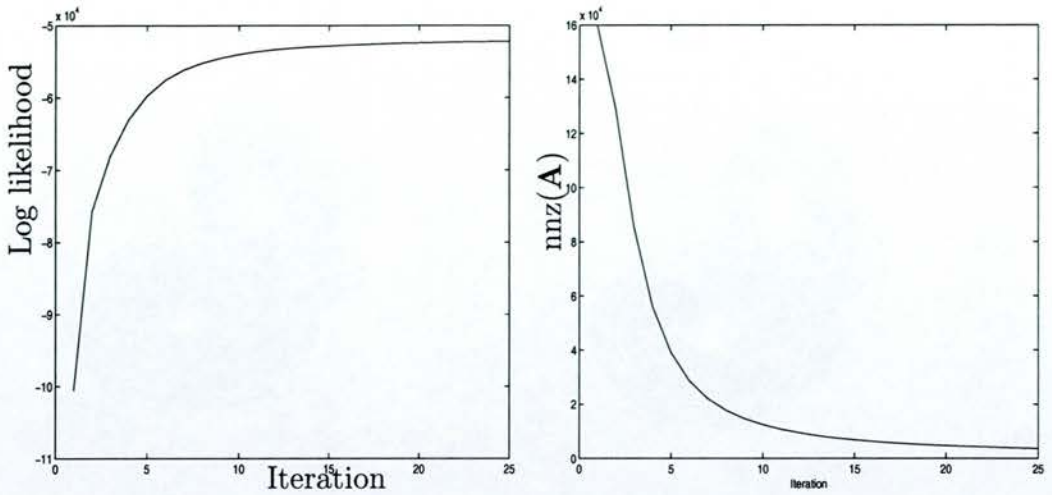
Figure 4.9: Lorenz Strange Attractor data with and without added noise.

algorithm. The thresholding "pruning strategy" outlined in Section 4.2.5.3 was applied after each iteration was complete.

The progress of the training is shown in Figure 4.10. As can be seen, (Figure 4.10(a)), the log likelihood steadily increases and has levelled off to a roughly constant value in the 25 iterations. While the log likelihood is still rising slowly, it has been found in practice that once this kind of plateau has been reached, that further training makes only very slight differences to the results obtained. Figure 4.10(b) illustrates the rapidly developing sparsity in the transition matrix as training proceeds. The initial probability matrix was dense, and so had 160,000 parameters, but as can be seen this rapidly declines, in line with the discussion in Section 4.2.4.3.

4.3.1.1 Visualization of the transition matrix

The developing sparsity pattern of the matrix is illustrated in Figure 4.11, where the elements of the 400×400 matrix have been displayed as an image in pseudo-colour coding. The probabilities were raised to the power 0.1 before forming the image, in order to compress the dynamic range of the displayed image and show more detail. Note that not only does the matrix develop

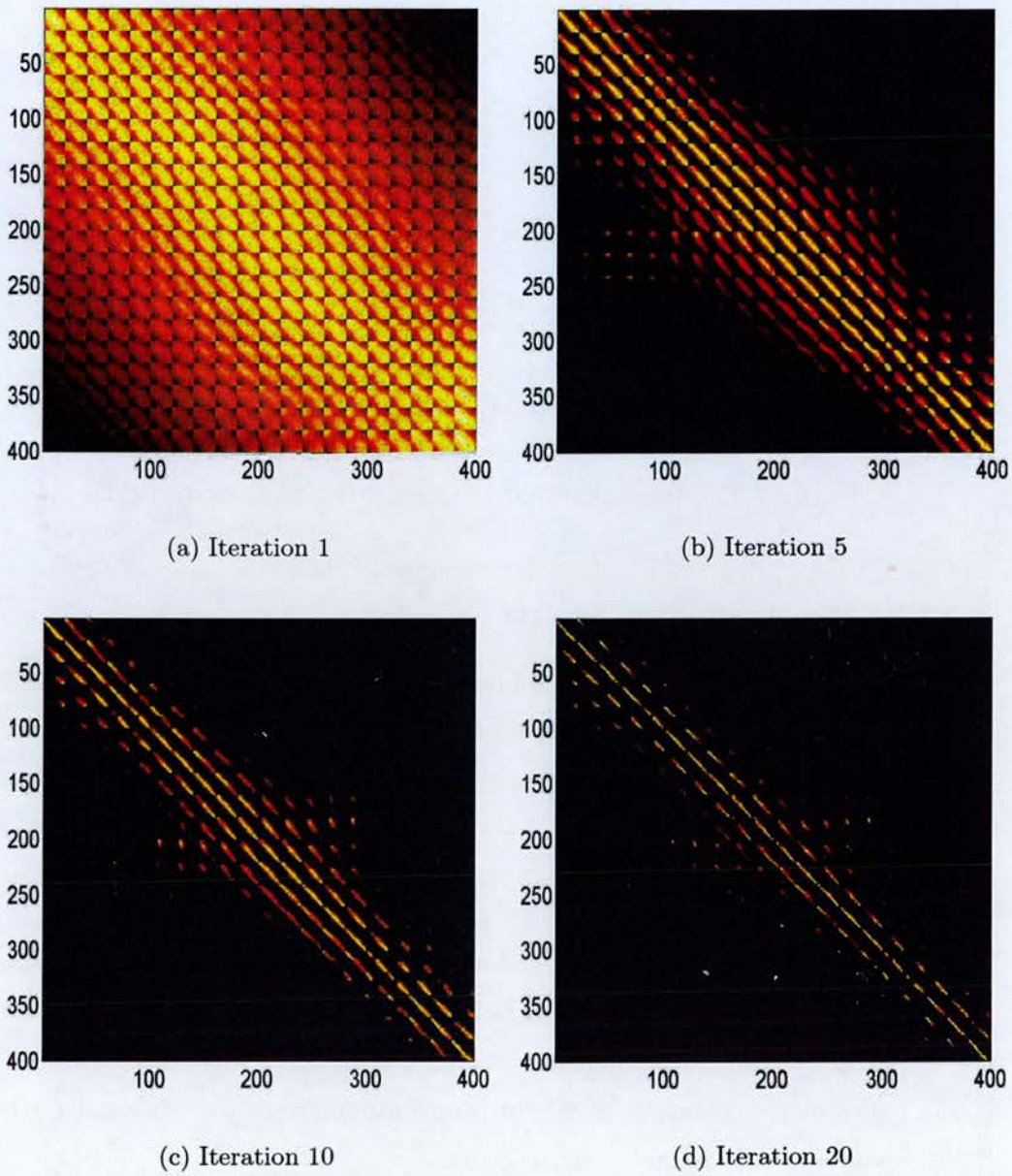


(a) Likelihood (b) Non-zeros in transition matrix

Figure 4.10: Curves of log likelihood and number of zeros in transition matrix for noisy Lorenz example during training

a sparse structre, but that a marked asymmetry also arises, particularly in the strongly off diagonal elements in Figure 4.11(d), which correspond to the times when the dynamics are changing rapidly between the two characteristic "eddies" in different regions of data space. We expect the matrix to be asymmetric, because the underlying data, being a dynamical system, has a strong directional flow.

This is further illustrated by representing the transition matrix as a vector field. This is illustrated in Figure 4.12, where the direction and magnitude for each vector displayed is calculated from the expected position of the next data point, using the transition matrix to update the position in the absence of a measurement. In other words, given the grid location $\mathbf{x}_j$, we compute $\sum_i A_{ij}\mathbf{x}_i$. The vector field shows that the model has captured the circulating dynamics, and also that the transitions between the two modes of oscillation are shown to take place at a region in latent space near the top of the diagram.



(a) Iteration 1

(b) Iteration 5

(c) Iteration 10

(d) Iteration 20

Figure 4.11: Pseudo-colour plots of the transition matrix probabilities during training for the Noisy Lorenz Strange Attractor example, at 1,5,10 and 20 iterations. Note the increasing sparsity, as the model learns the relevant transitions. In order to show more detail on the images, all probability values have been raised to the power 0.1, which compresses the dynamic range of the image. Note that the band structure in the off diagonal elements is in anti-phase (clearly visible after 20 iterations), because directional flow is being modelled and hence $A_ij \neq A_ji$.

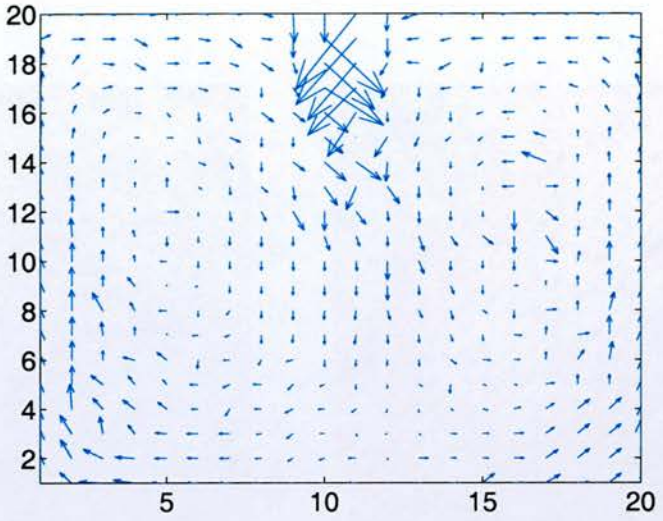


Figure 4.12: Transition matrix represented as a vector field.

4.3.1.2 Visualization of the data

Visualization of the data may be achieved by plotting an appropriate statistic from the posterior distribution in latent space. This could be done by computing the mean of the emission density, which corresponds to what would be done for a standard GTM. However, having taken temporal context into account with GTM Through Time, we have two options available:

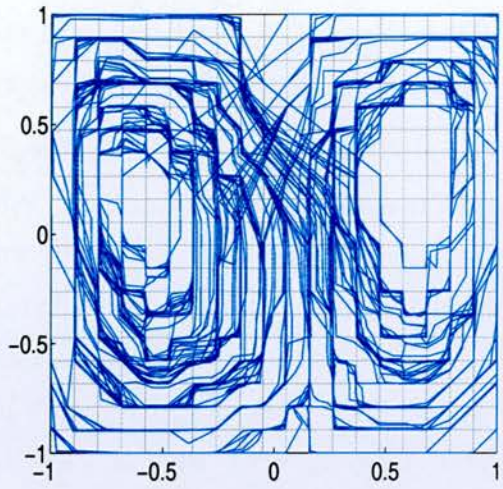
1. Plot the mean of the $\boldsymbol{\gamma}_t$ distributions from the Hidden Markov Model, which give the probability of being in state X_i at time t, and are computed by a single pass through the Forward–Backward algorithm. This will give the greatest degree of temporal context, as it takes the whole observation sequence into account.

2. Plot the mean of the $\boldsymbol{\alpha}_t$ distributions (the Forward variables), which only take into account the past time history. This approach would be essential for on-line analysis of data, if results are to be displayed immediately after the reading is taken. If a delay of one sequence-length is acceptable, then the Forward–Backward algorithm could be applied in real time, given sufficient computer power. However, the forward only option will also be less computationally demanding. There are two problems to be aware of when this is done:

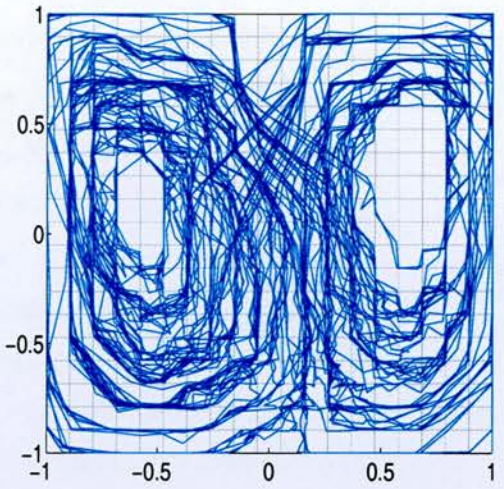
(a) Cumulative rounding error can lead to the collapse of the distribution for very long processing sequences, so it is advisable to chop the data up into short sequences, reinitilizing with the prior $\boldsymbol{\pi}$.

(b) There is a transient effect associated with the above step, because the first point in each sequence is the mean of the prior, irrespective of the previous readings, which are not taken into account. In the forward–backward algorithm, this would be smoothed out by the future temporal context. This effect may be overcome by processing overlapping sequences of data vectors; if we have just processed the sequence $(\mathbf{x}_t; t = t_0 \ldots t_0 + N)$, then the next sequence processed is $(\mathbf{x}_t; t = t_0 + N - k \ldots t_0 + 2N - k)$, where k is the overlap, of a few data points. The first k points of the new sequence are then discarded. Thus a computational overhead will be involved during the time of the overlap, because two models will be running in parallel, one for the end of the last sequence, and one for the beginning of the next sequence, during its transient settling time.

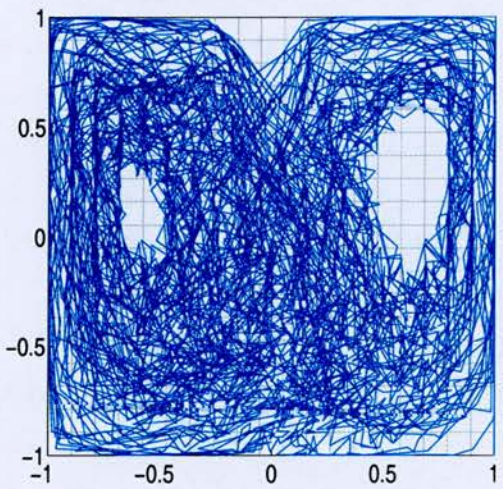
The visualization of 5000 of the points in latent space is illustrated in Figure 4.13, for the two temporally dependent methods (Figure 4.13(a), and 4.13(b)), and for comparison, using the emission probabilities from the standard GTM model (Figure 4.13(c)). The figures illustrate that the visualization achieved with the forward–backward algorithm exhibits the greatest degree of temporal smoothing, as it takes into account the full temporal context of the data sequence, using past and future context. The visualization using just the forward probabilities is noticeably "noisier", but nonetheless clearly shows the temporal evolution characteristic of the Lorenz Attractor. By contrast, when just the emission probability is used, which does not use temporal context, there is evidently no smoothing, resulting in a more cluttered visualization, which makes it more difficult to discern the underlying dynamical processes. For clarity, in Figure 4.13(d), only the first 1000 points have been plotted. Again it is clear that without temporal context, the noise, which is uncorrelated with time, has not been effectively filtered out. This plot was obtained by treating a GTM through time model as a standard



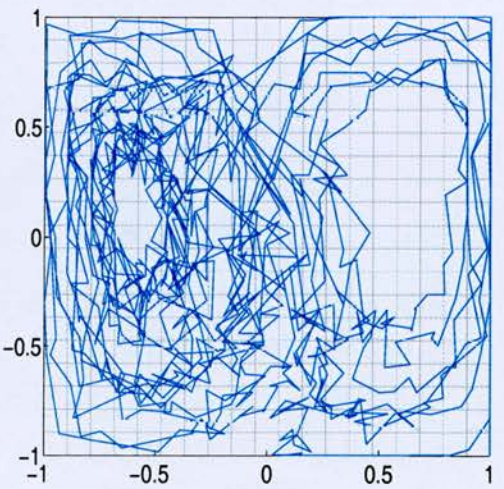
(a) Forward-Backward probabilities $P(q_t = X_i|\mathbf{O}, \lambda)$



(b) Forward probabilities $P(q_t = X_i|(\mathbf{y}_1 \ldots \mathbf{y}_t, \lambda)$



(c) Emission probabilities $P(q_t = X_i|O_t)$



(d) Emission probabilities first 1000 points

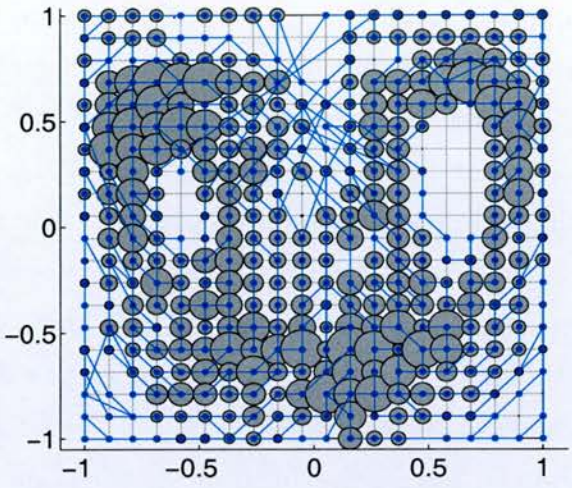
Figure 4.13: Visualization of 5000 points of the Noisy Lorenz data in latent space by computing the means of the forward, forward–backward and emission probability distributions. In (a), the whole sequence is taken into account, thereby achieving the the greatest degree of temporal smoothing. In (b), only the past context is used, resulting in a slightly less well-smoothed visualization. In (c), there is no temporal context, and hence no smoothing can be achieved. Plot (d) shows only the first 1000 points for clarity.

GTM, though it was trained with the time dependent algorithm. A control test was also performed using the standard GTM training algorithm. Here, the non-linear mapping produced was virtually identical to that of the GTM through time model, but the inverse variance parameter β had a different final value, being 0.474 for the standard GTM, and 0.678 for the GTM through time. This implies that the GTM model produced a greater variance than GTM through time. However, the extra variance, which would be due to the effect of noise in the data set, did not enable the model to smooth out the noise when posterior means were computed, and the corresponding plot to Figure 4.13(c) using the straight GTM model was very similar to the emission density plot from GTM through time.

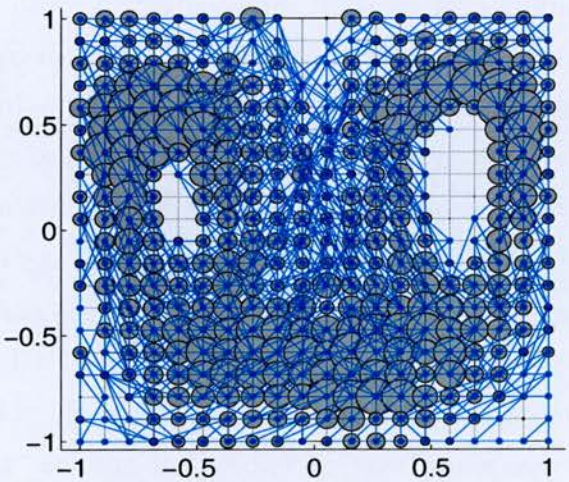
In Figure 4.14, an alternative form of visualization of the data is presented, with and without temporal context. This was produced by computing the most probable system state at each time step and plotting the data at the corresponding grid location. Since this will mean that many different data points are superimposed on each grid point, an indication is given of how many visits there have been to a particular grid point by plotting a grey circle, centred on each grid point, whose radius is proportional to the total number of visits made to that point for the whole data set.

In the case of Figure 4.14(a), the visualization was computed using the standard Viterbi decoding algorithm for a Hidden Markov Model. This algorithm is based on dynamic programming methods. It involves a forward pass through the data, where for each state X_j at time t, we find the X_i at time $t-1$ that maximized the probability of arriving at state X_j (i.e. the best state sequence to time t ending on state X_j). At the end of the forward pass we have the most probable state at the end of the sequence, and then back track through the previously computed quantities to compute the best path for the whole sequence. Details of the algorithm are given in Appendix B.

In the case of Figure 4.14(b), the most probable state is computed without the benefit of temporal context, and thus corresponds to the mode of the emission density. This is analogous to the "winning node", in a SOM. The plots again show that taking temporal context into account has produced a less cluttered trajectory, making it easier to see the underlying processes.



(a) Viterbi



(b) Emission density mode ("winning node")

Figure 4.14: Visualization of data using the most probable state sequence, computed in (a) using the standard Viterbi decoding algorithm, and in (b) using the mode of the emission density, which is analogous to the "winning node" in a SOM. Visualization points are restricted to the grid locations corresponding to the states. Grey circles have a radius proportional to the number of visits to the corresponding location.

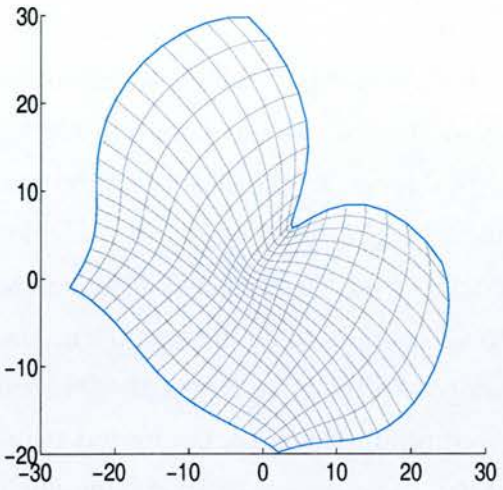
4.3.1.3 Visualization of the GTM manifold using NeuroScale

One known drawback of non-linear latent space methods based on density models is that the flat latent space visualization tends to smear out the details of the distribution. We have already discussed this in Chapter 2 in the context of the Kohonen Map, which, while not a latent space model, still attempts to fit an L dimensional manifold through the data, and thus suffers from similar problems. In the case of a mixture model like this, it will place more nodes in the regions of high density, and less in the regions of low density. When "flattened out" into a 2-D visualization, this information is lost, and the distribution tends to be more uniform. In [Bishop et al., 1997b], this problem was addressed by using **Magnification Factors**, which are the degree of local stretching on the map, computed using using the techniques of differential geometry. The degree of stretching could then be represented, for example by pseudo-colour coding.

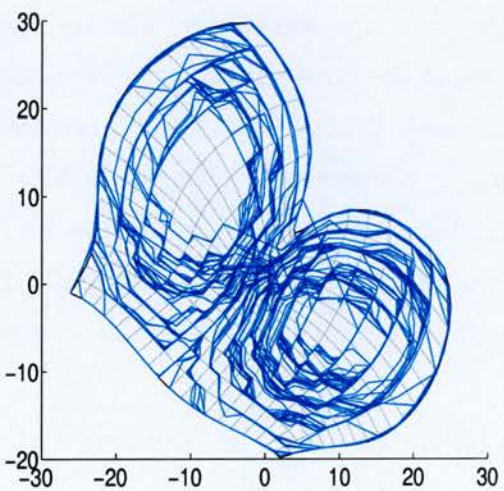
Here, we introduce a different technique, which can be used to give a visualization of the GTM manifold, and the data simultaneously. The technique is simply to re-map the data locations using the NEUROSCALE topographical mapping reviewed in Chapter 2 of this thesis. NEUROSCALE is a functional mapping that has the topology preserving properties of a Sammon Map, in that it attempts to preserve inter-point distances in visualization space. The Sammon Map has the disadvantage that all the data has to be trained at once, but NEUROSCALE produces an RBF neural network whose mapping can be computed on new data.

We train the NEUROSCALE map on the locations in data space of the mixture component centres, and initialize the RBF centres with the locations in data space of the RBF basis function centres, initializing the NEUROSCALE basis function centres with the projections into data space of the GTM basis function centres.

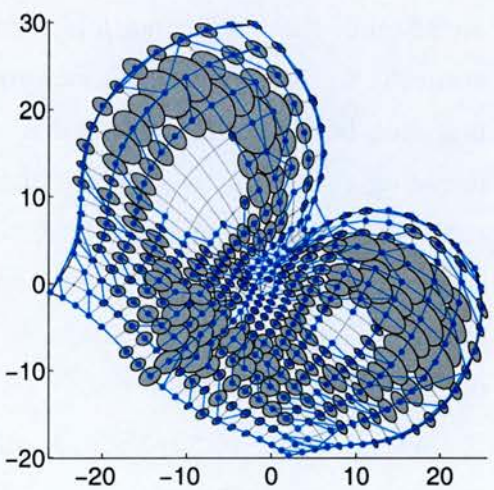
This is illustrated in Figure 4.15, where the centre locations only are plotted in Figure 4.15(a), and two of the visualizations discussed so far, namely the posterior means using the forward–backward algorithm (Figure 4.15(b)), and the most probable state path using the Viterbi algorithm (Figure 4.15(c)).



(a) GTM manifold



(b) Posterior means



(c) Viterbi

Figure 4.15: Visualization of the GTM mainfold, posterior means, and optimized state path, using a non-linear NEUROSCALE projection, trained on the data space projections of the latent space grid points.

The non-linear projection of the mesh itself shows that many grid points have been placed in the region where the transitions between modes occur. This aspect was not apparent on the flat mapping, which would have given the impression of more rapid transitions of the data than was in fact the case.

4.3.2 State-switching two-link robot manipulator

This example is intended to illustrate the capability of the GTM through time model to learn the characteristics of a temporal data set where non-local and local transitions can occur at successive time points, while still retaining temporal coherence between points.

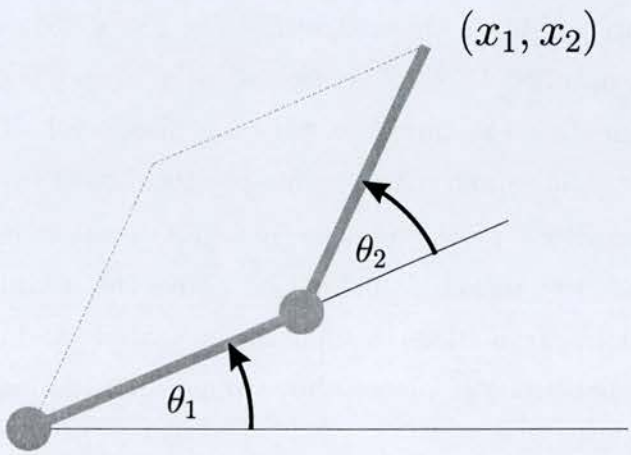


Figure 4.16: Schematic of Robot manipulator with alternative solutions, corresponding to "elbow up" and "elbow down" configurations.

Figure 4.16 is a schematic diagram of a two-link robot manipulator, illustrating that two different configurations of the robot (known as "elbow up" and "elbow down"), give rise to the same position of the end effector in Cartesian space. Data are generated by performing a random walk in joint space and computing the resultant joint position. Additionally, at each time step, the configuration may switch configuration from "elbow up" to "elbow down", with a fixed probability p. This is of course not a realistic scenario; it is provided so that examples of both local and non-local transitions may take place in the temporal sequence. For full details, see Appendix A.

For the purpose of this study, data sets of 100 sequences of 100 vectors were generated according to the above equations, for $p = (0, 0.02, 0.2, 1)$. Each sequence was initialized with θ_1 drawn from a normal distribution with mean $\pm\frac{\pi}{6}$. The positive or negative value of the mean was chosen at random. The three data sets look identical if the data are projected using Principal Components Analysis.

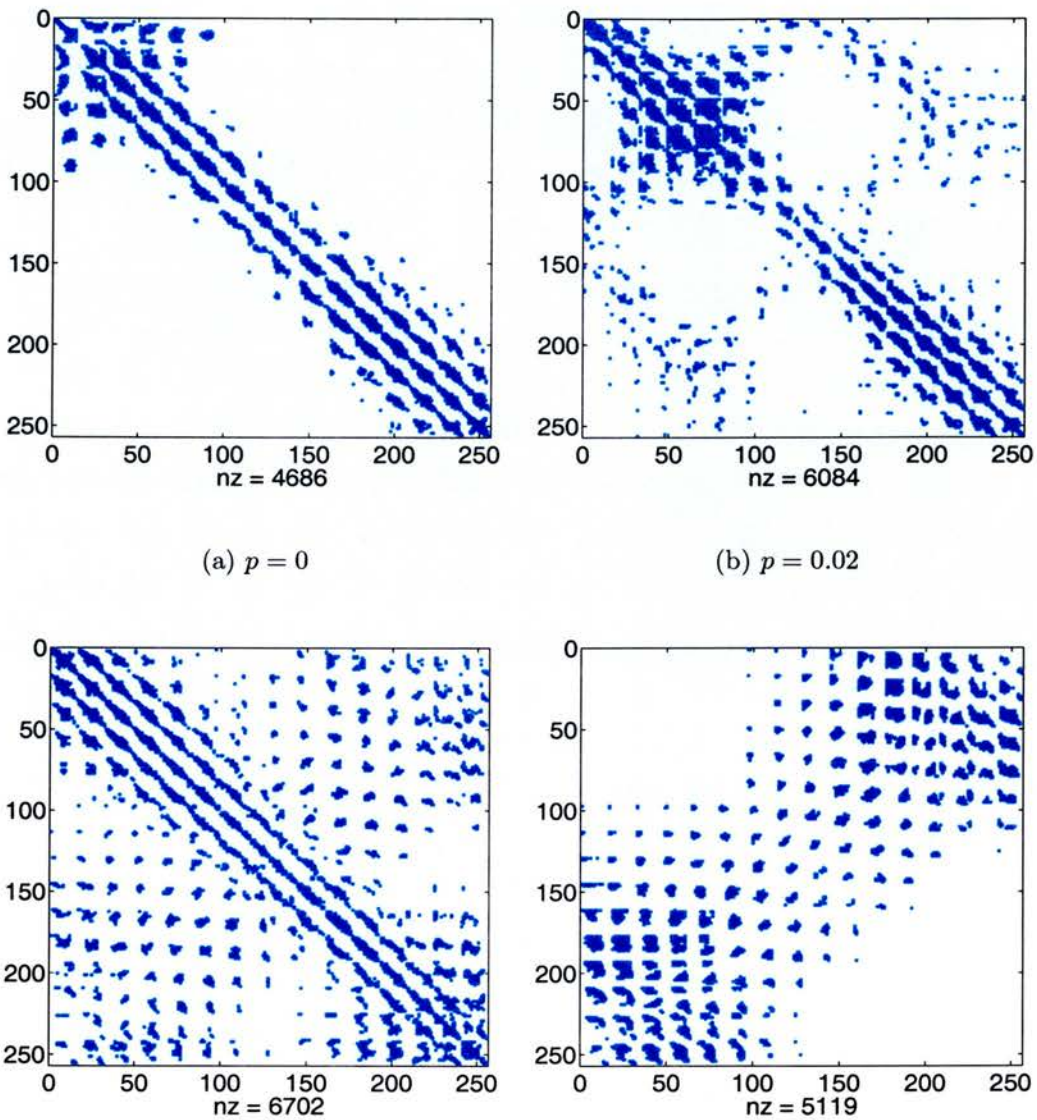
4.3.2.1 Sparsity Patterns learned by the model

Figure 4.17 illustrates the sparsity patterns of the transition matrix learned by the model for the four different values of the state switching probability p. The transition matrix relates to transitions between states represented by a 16×16 matrix. Hence the matrix itself is 256×256, where the i^{th} row represents an unrolled 16×16 square, whose j^{th} entry is the probability of transition from state i at time t to state j at time $t + 1$. The corresponding row and column for latent space is thus $\{\lfloor i/16 \rfloor, i \bmod 16\}$.

Hence transitions to the next or previous position in a row in latent space correspond to diagonals below and above the leading diagonal in the transition matrix. Transitions to adjacent rows (above and below) correspond to diagonals that are $\sqrt{K}$ places above and below the leading diagonal (in this case 16 places).

Thus a transition matrix where only localized transitions in latent space are allowed will have a band diagonal structure, with the distance between band centres equal to $\sqrt{K}$. By contrast, if non-local transitions are allowed, we should expect to see many entries that are far off the leading diagonal (or near the leading diagonal with a column number whose modulo 16 value is near 8, indicating a large lateral transition.

This effect is illustrated in Figure 4.17. In Figure 4.17(a), the transition probability is zero, and so the figure exhibits the banded diagonal format expected. Figure 4.17(b) (corresponding to $p = 0.02$) was generated by a system showing occasional transitions between elbow up and elbow down configurations (around every 50 time steps), but for the most part having local transitions. Thus the band structure is still strong, but off-diagonal elements appear in the matrix, indicating that non-local transitions take



(a) $p = 0$ (b) $p = 0.02$

(c) $p = 0.2$ (d) $p = 1$

Figure 4.17: Sparsity patterns developed by GTM through time on the state switching robot example, for different values of the switching probability

place, and are modelled correctly. Note also that because of the two different types of transition modelled, that the number of non-zeros has risen. Figure 4.17(c) was generated with $p = 0.2$, indicating relatively frequent transitions, around every 4 or 5 time steps, with the remainder local. As can be seen, the block diagonal structure is less heavily emphasized, and more off diagonal elements appear. Finally, in Figure 4.17(d), p is set to unity, implying a state switch on every time step. Thus virtually all the transitions are non-local (except in the case where $\theta_2 \approx 0$). This is reflected in the diagram, where the structure exhibited is in the opposite direction to the leading diagonal.

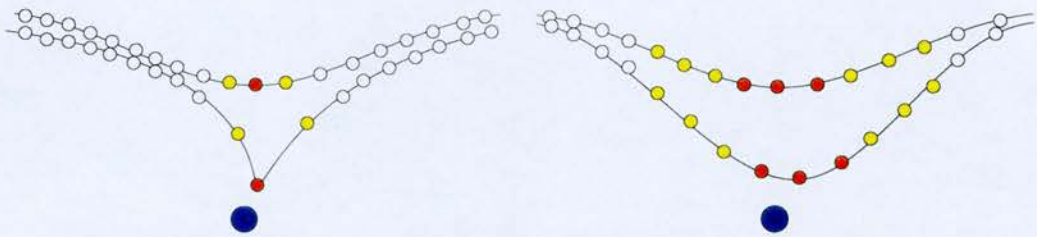
It should also be noted that the structure tends to be symmetric. This is because of the nature of the data-set, where each step is in a randomly selected direction in joint angle space; there is no preference for one direction over another. By contrast, the sparsity matrix of the Lorenz example shows asymmetry, as the direction of movement at any given point in state space is fixed.

4.3.2.2 GTM mapping learned by the model

Figure 4.19 shows the GTM manifolds learned for two of the switching state robot examples, and compares the meshes with those obtained from training a standard GTM with the same data and parameters. Figures 4.19(a) and 4.19(c) show the meshes obtained for the standard GTM models, for the $p = 0$ and $p = 1$, and figures 4.19(b) and 4.19(d) are the corresponding meshes obtained for GTM through time with the same data sets. In all cases, the same non-linear projection mapping was used, in order to compare like with like.

We should expect that GTM through time would produce a different kind of manifold to static GTM, because the update equations for the $\mathbf{W}$ matrix that determines the linear combinations of GTM basis functions used in the mapping, is updated using the posterior probabilities from the Hidden Markov Model, rather than the standard GTM responsibilities. As discussed earlier, we have found that these will give rise to much more tightly clustered distributions, because the temporal context enables the position in latent space to be estimated more accurately.

The effect of this that we expect to see is that the GTMTT mappings will exhibit greater curvature than those of the GTM. This is illustrated schematically in Figure 4.18.



(a) Localized responsibility (b) Spread out responsibility

Figure 4.18: The difference between localized and non-localized responsibilities on the learning in an EM algorithm (see text for explanation).

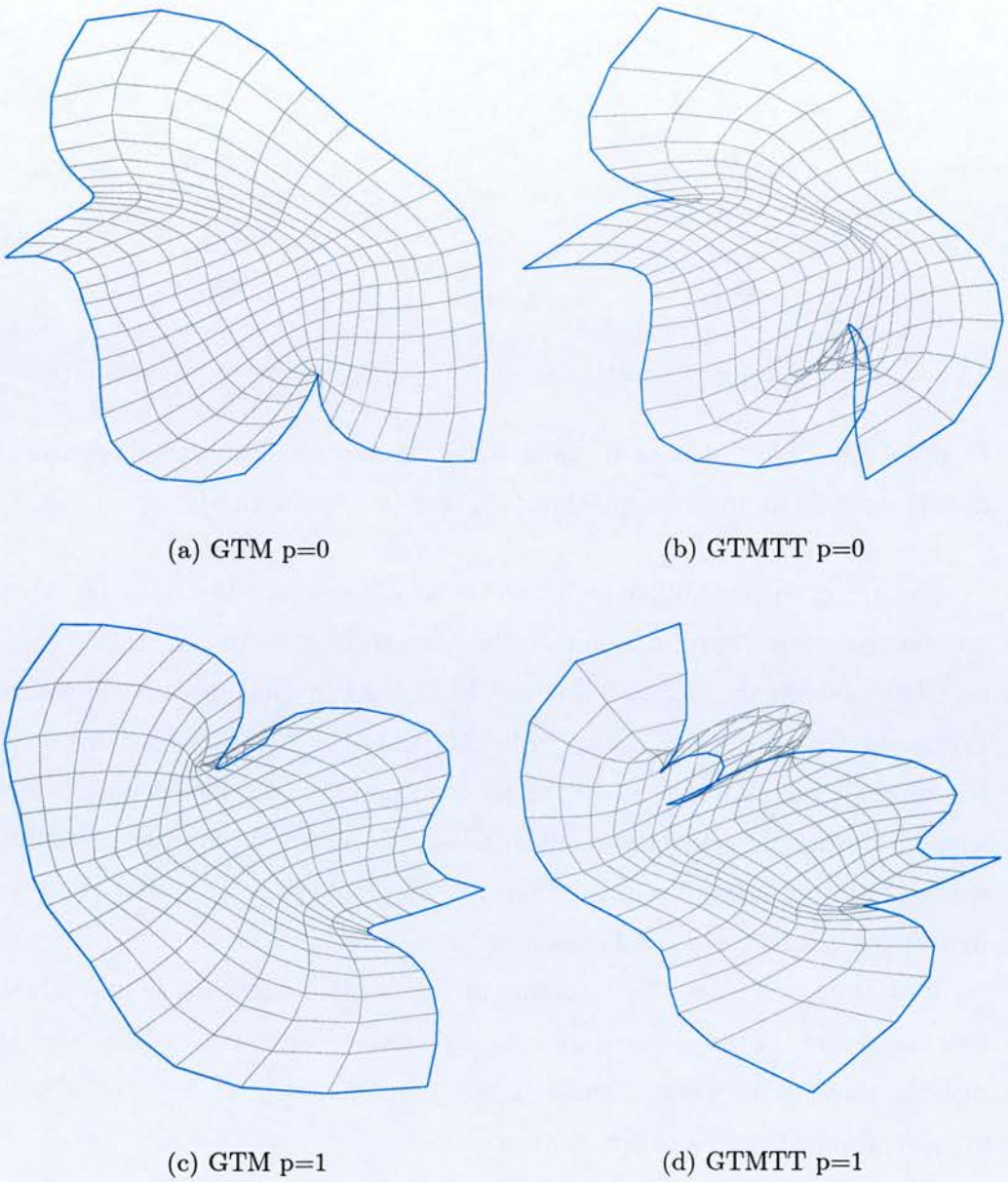
Assigning responsibility to a node draws it towards the data point, with an effective "force" proportional to the responsibility[3]. In GTM, there is also an effective restraining force, due to the interaction between non-linear basis functions, that keeps the mapping locally smooth. A sharply localized set of responsibilities, as found when taking temporal context into account, means a much stronger force on a small number of centres. If the responsibility is spread out over a large region of centres, the force is gentler, and the overall distortion is less, resulting in a smoother manifold.

It is apparent from the diagrams in Figure 4.19 that this is just what we see; the shapes of the meshes are roughly similar, but the GTM through time meshes show more violent kinks in the map, possibly thereby exhibiting a greater ability to explore the data set.

4.3.2.3 Path visualization using the model

Visualizations of two separate vector sequences are shown in Figure 4.20, using the Viterbi algorithm to compute the most probable state sequence. The diameter of the grey circle on each grid point is proportional to the

[3]In contrast with the classical k-Means algorithm, where *all* the responsibility is assigned to a single centre, and only that centre moves, in an EM algorithm, the responsibility is spread out, so potentially all the centres move; those with most of the responsibility move the most.



(a) GTM p=0

(b) GTMTT p=0

(c) GTM p=1

(d) GTMTT p=1

Figure 4.19: Comparison of non-linear manifolds learned by GTM and GTM through time for the switching state robot example, for the $p = 0$ and $p = 1$ cases. GTM through time produces mappings with sharper features due to the more localized posterior distributions (responsibilities) that arise when temporal context is taken into account.

number of visits to that particular node, taken over the whole data set, and is thus a crude indicator of the probability density. The red and green traces are individual vector sequences. The increasing numbers of transitions are clearly visible from the four different maps, illustrating that, if necessary, GTM through time can model local transitions, non-local transitions, and a combination of both.

Note also that the mappings learnt differ for the different values of the switching parameters. This contrasts with a standard PCA model, where the visualizatons of the four data sets are essentially the same.

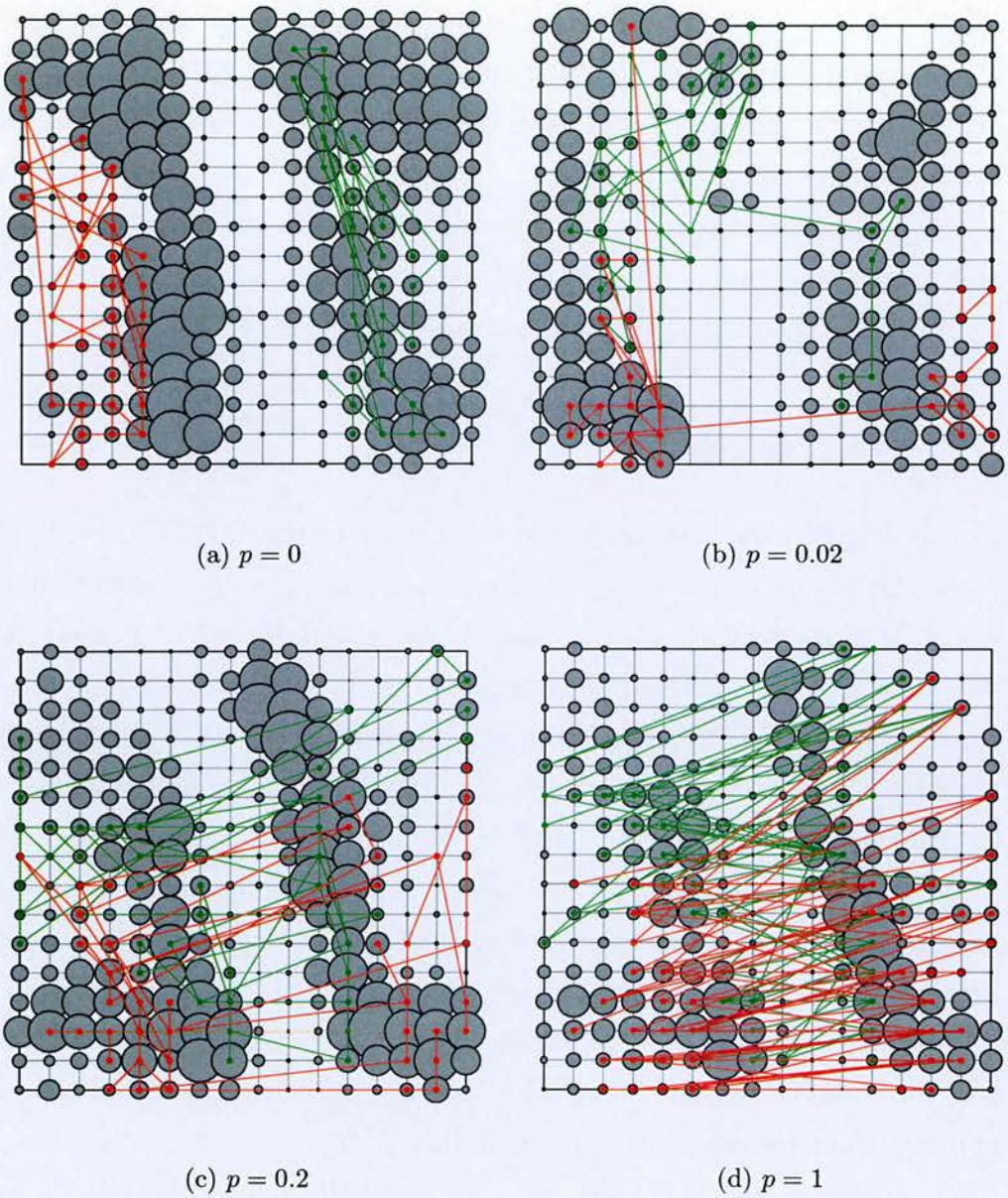
4.4 Discussion

In this chapter, we have presented the theory of the GTM Through Time model for visualization of high dimensional data. We have demonstrated that it is a practical algorithm, that can be straightforwardly implemented. While its processing and storage requirements are more demanding compared to the standard GTM model, they are not excessively so, and furthermore considerable gains may be made in both processing and storage requirements by utilizing sparse representations.

We expected that the value of using temporal context processing would be in the ability of the model to filter out measurement noise, and give a better visualization of the underlying processes. This was demonstrated by the use of two examples, one based on the Lorenz Attractor, with a high amount of added measurement noise, and one based on the Kinematics of a hypothetical two-link robotic manipulator.

It was found that GTM through time could effectively filter out the noise in the data, and thereby give a clearer representation of the underlying processes taking place than is possible with the standard GTM model . It was further demonstrated that GTM could learn arbitrary transition matrices, where sudden large transitions could take place over a single time-step.

We also demonstrated that the actual mapping learned by GTM Through Time (the non-linear manifold produced by the GTM's RBF mapping) also differed from the static case, in a manner that was expected. The difference appears to be that GTM through time can produce a more complex map,



(a) $p = 0$ (b) $p = 0.02$

(c) $p = 0.2$ (d) $p = 1$

Figure 4.20: Visualizaton of robot paths in latent space using the Viterbi algorithm, for no switching (a), occasional switching (b), frequent switching (c), and constant switching (d). The grey circles show the number of visits to each grid point for ALL the data, the size of circle scaling according to the number of visits. Green and red traces are selected individual sequences.

that more effectively represents the data set. By contrast, because GTM is unable to filter out noise, it therefore has to develop a smoother manifold, where more of the data set is put down to noise. The sharper features of the GTM Through Time map are explainable in terms of the more localized posterior distributions that arise in the training when temporal context is taken into account.

GTM Through Time is a computationally demanding algorithm, but many economies may be introduced by exploiting the sparse nature of the transition matrix that develops during training. As has been demonstrated, it is feasible to start with a relatively small model, to learn a sparse representation, and then scaling up the resolution of the mapping.

The principal drawback of the algorithm, which is common to all such techniques were a flexible manifold is fitted through the data, is that the manifold can fold over in regions where there is a high density of data points, in order to produce a better probabilistic model. However, since the folding over produces multimodal distributions in latent space, this makes the (subjective) visualization harder to interpret. What this means in practice is that it is not a particularly beneficial technique to compare likelihoods between models with different values of the definable parameters, in order to get the best visualization. In addition to the likelihood values, other subjective features enter, such as the ease of interpretation of the visualization.

In order to address the problem of folding over, we have introduced the idea of using a NEUROSCALE map, trained on the manifold grid points, in order to produce a mapping that combines the probabilistic techniques and temporal smoothing of GTMTT with the desirable topology preserving properties of NEUROSCALE .

Chapter 5

Application to Patient Monitoring Data

5.1 Introduction

In this chapter we apply the techniques developed in the earlier chapters to a large data set. Specifically, we shall be analyzing Patient Monitoring data, that has been collected on custom-developed hardware by the Oxford University Engineering Science Department (Signal Processing And Neural Networks Group).

In this application, hardware has been developed for monitoring patients in Intensive Care, High-Dependency Care and Coronary Care by sampling a number of parameters, which are stored on a computer hard disk. Ongoing research involves developing analysis methods for this data to assist in monitoring of patient status.

There are six data sets in all, corresponding to different patients who were monitored using the apparatus. During this time, two of the patients showed "trajectories" in their state, in one case from an unstable condition to a quasi-stable state, and in the other case from quasi-stability to an unstable state. The objective of this study is to see if the visualization techniques developed in the thesis can be applied successfully to this data, and in particular to see if the trajectories can be visualized in two dimensions.

The rest of this chapter is divided up as follows:

- In Section 5.2, the raw data is described.
- In Section 5.3, we describe the application of Probabilistic Principal Components Through Time to the data of one of the patients.
- In Section 5.4, we present the results of training models on single patients.
- In Section 5.5, we present the results of training models on the entire group of patients, so that the monitoring of individual patients can be seen in the background of the ensemble of patients.
- In Section 5.6 we discuss the results obtained, and attempt to assess the merits of the temporal approach.

5.2 Description of the data

The data sets were acquired using the "Software Monitor", a portable PC that records vital physiological signals from hospital patients [Tarassenko et al., 2001]. In all, six data sets were used, obtained by monitoring various patients in a Coronary Care Unit, for periods of time varying from approximately 7 to 23 hours. Four variables were monitored, at different sampling rates. The six data sets used in this study were supplied normalized from statistics gathered from a much larger base of around 80 patients. The offsets and scaling factors used for the normalization were such that the entire database of 80 patients would have zero mean and unit standard deviation for each of the signals. It is noted in the above cited paper that the idea of normalization based on prior knowledge had also been considered. This would involve clinical experts making judgements about the relative importance of the features in the patient groups, for example, if a 0.5° C change in skin temperature equivalent to a 50 mm Hg change in blood pressure (in terms of the condition of the patient). However these normalisations were not available for this study.

Table 5.1 shows the sampling intervals for each variable, together with the mean and variance over the 80 patient data set.

Variable	Sampling Interval	Mean	Variance
Heart rate (bpm)	2s	90.3	397.3
Systolic blood pressure (mm Hg)	15 min	128.73	579.5
O_2 saturation (percent)	2s	94.88	13.62
Temperature (Celsius)	2s	34.65	4.852

Table 5.1: Patient Monitoring signals - basic statistics

In order to produce a sequence of data vectors that are all 4 dimensional, the systolic blood pressure measurements have been linearly interpolated between measurement readings. In principle, it should be possible in GTM and GTMTT to cope with different sample rates for different variables by probabilistic techniques. Since the emission density is a probability distribution, for data points where some variables are not measured, one can still estimate the density by integrating over the unknown variable.

5.2.1 Previous work on the Patient Monitoring Data

Existing work on data from the software monitor has concentrated on two areas, that of intelligent signal processing using Data Fusion techniques, and data visualization [Tarassenko et al., 2001]. Data Fusion is important in this kind of application, to distinguish between artefact (for example interruption of, or excessive noise on a sensor signal), and real medical events that need to be detected (such as ectopic heart beats).

In one particular unpublished experiment[1], data from a particular patient who progressed from a quasi-stable to an unstable state, was processed by learning Kalman Filter dynamics, using software that has been developed for Jet Engine Condition monitoring [Utete et al., 2000]. In this experiment, the linear dynamical system was trained using the first 10,000 data points, during which the patient was in a stable state, and then the model was used to process the rest of the data. The deterioration of the patient was visible through the Kalman Filter Innovation term; a measure of the difference between the predicted data values from the Kalman Filter and the measured values. This particular experiment did not involve visualization; the dimen-

[1]L. Tarassenko: Private communication

sion of the state space was equal to the number of variables being monitored. In section 5.3 we essentially replicate this experiment, but also attempt to combine it with data visualization, by using PPCA Through Time.

Data visualization techniques described in [Tarassenko et al., 2001] utilized the NEUROSCALE technique described in Section 2.2.3, in two different ways. In the first case, the NeuroScale map was trained solely on the data from a single patient, and then the interpolating properties of NEUROSCALE were used to generate trajectories in the 2-D map over the monitoring period. In the second case, a NEUROSCALE map was trained on data from *all* patients in a particular group, and then the map was used to compare one patient with another. In both cases, the $K-$means algorithm was used to pre-cluster the data, to reduce the number of data points used in training NEUROSCALE to a tractable size (because for N data points, the $N(N-1)/2$ Euclidean distances to be calculated).

We follow essentially the same procedure in this study with GTM Through Time, using a single patient's data for trajectory calculation (Section 5.4), and comparing an individual patient against the ensemble of patients (Section 5.5). We also use NEUROSCALE in the study, but our approach differs from Tarassenko et. al, in that we use the learned GTM basis function centres as the training patterns for the map, as opposed to pre-clustered feature vectors. This is the technique discussed in Section 4.3.1.3.

5.3 Experiment with PPCA Through Time

The objective of this isolated experiment was to see if a simple visualization technique such as PPCA Through Time, as presented in Chapter 3 of the thesis can be used to assist in the visualization of the data for this application, and in particular to see if the probabilistic framework can give a detection of a change in status during a monitoring period.

For this example, the monitoring data from "Patient 56", was chosen, which is a case where the patient progressed from a quasi-stable status to an unstable status.

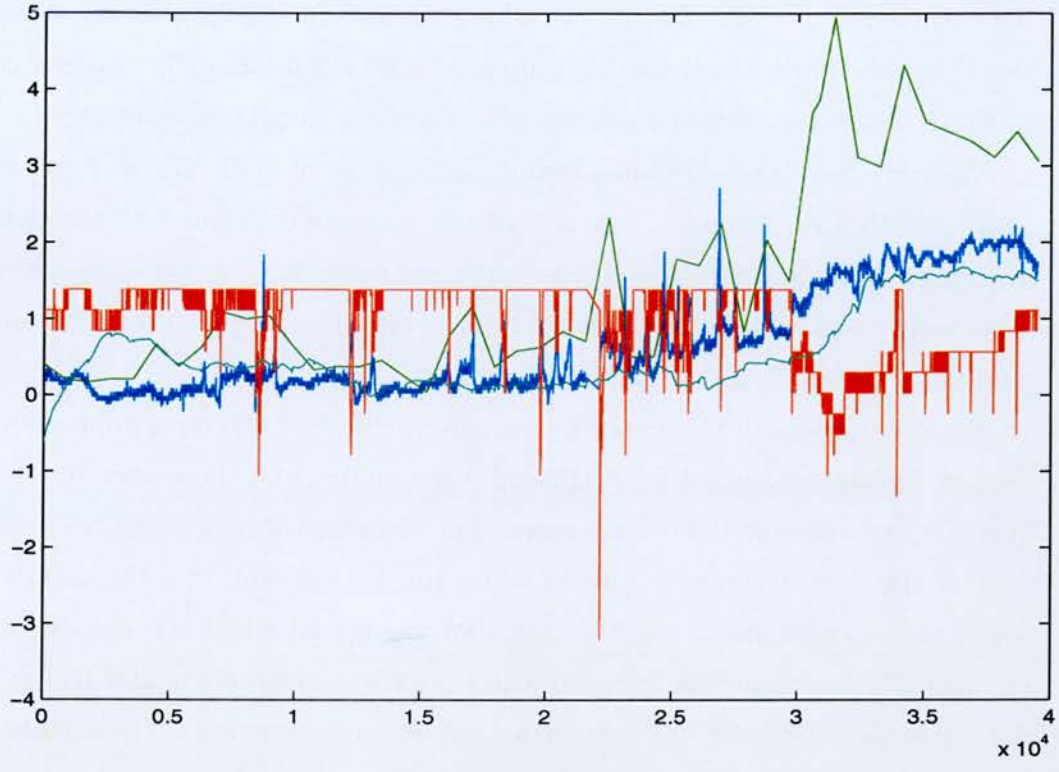


Figure 5.1: Normalized monitoring data for "Patient 56". The four variables measured are **Heart Rate** (blue trace), **Systolic Blood Pressure** (green trace), O_2 **saturation**, (red trace) and **Temperature** (turquoise trace).

The measured data for this patient is shown in Figure 5.1, which represents measurements over a period of approximately 22 h, and sampled and preprocessed according to the scheme described in the last section.

It is apparent from examining Figure 5.1, that there is a change in the condition that begins to set in at around sample 20,000, but is most apparent from around 30,000 onwards, characterized by a simultaneous rise in temperature, heart rate and blood pressure, and a drop in O_2 saturation. Before that occurs, the condition could be described as "quasi-stable". The other notable feature of the graph is that during the early stages the O_2 saturation appears to have a cut-off. The raw figures for this measurement were given as percentages, with the cut-off being at 100 percent, and the mean being around 98 percent. The saturation at 100 percent for this patient is not due to any sensor failure, as it might seem from a simple inspection of the graph, but a result of the fact that the patient was wearing an oxygen mask during

the monitoring period. The points where the saturation level drops off for short periods (particularly the big drop at around sample 22,000), correspond to times where the oxygen mask fell off, causing a drop O_2 saturation.

We note here that the underlying assumption of both GTM Through Time and PPCA Through Time is that the measurement noise on the data is normally distributed. This is clearly not the case with this variable, which is coarsely discretized, with a cut-off that results in a lop-sided distribution, so strictly this is an inappropriate model[2].

For the purpose of this experiment, only the first 10,000 data points were used in the training of a PPCA through time model, and these were divided into 100 sequences of 100 data vectors. The objective of only using the early part of the data (corresponding to when the patient was in a quasi-stable state) was to examine the performance of the model when the data being monitored was outside the range of the training data, in particular to see if the behaviour indicates that abnormal events are occurring. The simplest form of the model, with state space of size 2, was chosen:

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{b} + \mathbf{r} \tag{5.1}$$

$$\mathbf{y}_{t+1} = \mathbf{W}\mathbf{x}_t + \mathbf{s} \tag{5.2}$$

Once the model was trained, various ways of post-processing the data were attempted, using the full data set. The simplest form of post-processing consists of treating the data as one long sequence, and only using the forward pass of the Kalman Filter to estimate the state variables. This method would be suitable for on-line processing of the data, although it only takes into account the previous history, rather than the entire vector sequence. The results are shown in Figure 5.2, where every 10th monitoring point has been plotted. Data corresponding to the first 20,000 points are shown in blue, and the remainder are shown in red. The trajectory of the red trace shows clearly the large change that occurred during the latter part of the monitoring. The looped feature in the plot that has its apex at the approximate coordinates

[2]The choice of a Gaussian noise model, as noted wryly in [Roweis and Ghahramani, 1999] is strongly motivated by considerations of analytical tractability, and more weakly by an appeal to the Central Limit Theorem.

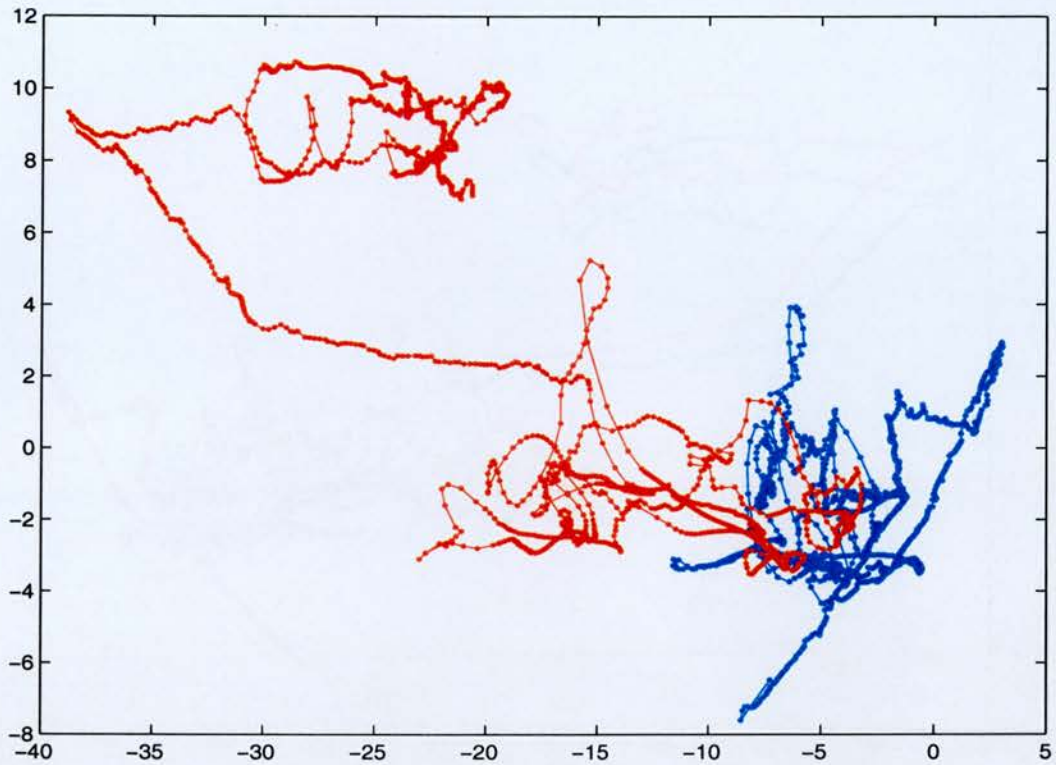


Figure 5.2: Two dimensional visualization of the data from "Patient 56", using the estimated state variables from the Kalman Filtering process. The first 20,000 points, are depicted in blue, and the remainder in red.

$(-15, 6)$ corresponds to the point in the monitoring described earlier, where the patient's oxygen mask fell off.

The effect of using Kalman Smoothing is shown in Figure 5.3. Here again, the data has been treated as one long sequence, and processed with a forward pass (Kalman Filter), and a reverse pass (Kalman Smoother) through the data sequence. The resultant estimated latent space trajectory is shown in red, with the Kalman Filter trace, identical to that of Figure 5.2, shown in blue for comparison. It is evident that the effect of the Kalman Smoother is to produce a much smoother plot. It is also apparent that as the monitoring proceeds to the novel data in the latter half of the set, that the agreement between the Kalman Smoother estimates of the latent variables and those of the Kalman Filter become increasingly divergent. This is because the effect of the novel data will be to produce a large correction term in the measurement update of the forward pass, from Equation (3.44). This results

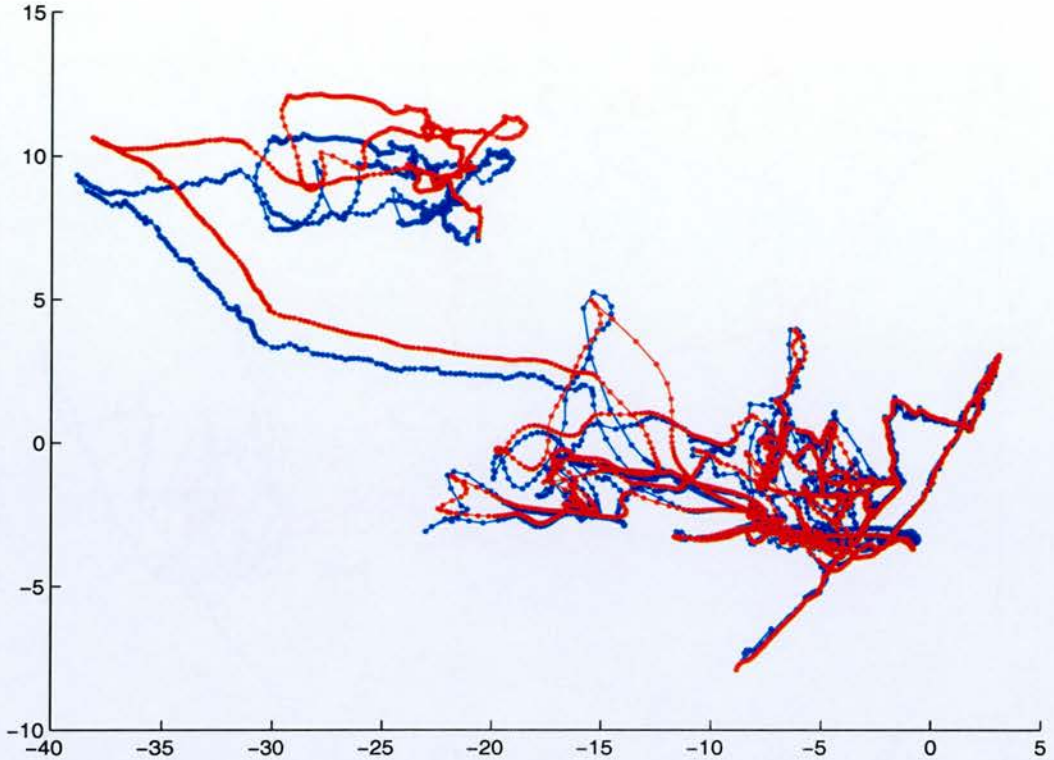


Figure 5.3: Comparison between Kalman Filtering (blue trace) and Kalman Smoothing (red trace) for the Patient 56 monitoring data.

in a drift in the latent space trajectory resulting from the linear dynamical response. The drift is partially cancelled out by the smoothing.

The resultant discrepancy between the filtered estimate and the smoothed estimate would be a good visual indicator of novelty, but it should be noted that this way of processing the data would be completely impractical for the application under consideration, as the results could only be computed after the entire 22 Hours worth of data had been logged. To achieve continuous monitoring, and at the same time exploit the smoothing effect of the reverse pass, it is necessary to break the data up into sequences, and process each sequence separately. This was performed on this data, where the 39,500 sample data set was broken up into 100 sequences of 395 vectors. The results are shown in Figure 5.4.

Once again, in the region of the training data, there is very good agreement between the two traces (the blue trace corresponding to treating the data as a single sequence, and the red trace corresponding to sequences of

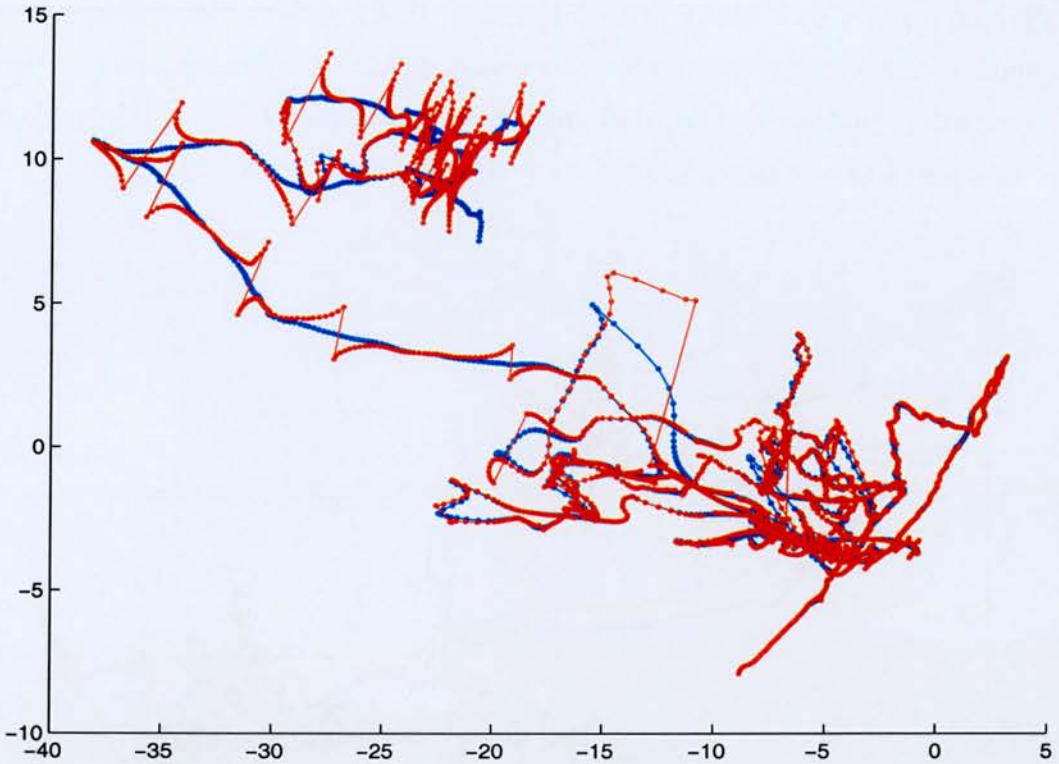


Figure 5.4: Comparison of Patient 56 data using full Kalman smoothing, with the data treated as one sequence of 39,500 vectors (blue trace) and 100 sequences of 395 vectors (red trace).

395 vectors). However, as the patient's condition deteriorates, the effect of breaking the data up into sequences becomes very noticeable, in that there are marked transients at the beginning and end points of each of the sequences. We should expect this to happen, as both the forward and reverse passes in the Kalman Smoothing algorithm involve linear dynamics, and will therefore exhibit transient settling behaviour. The agreement between the two is best in the middle of the sequences, where these effects have decayed away. The manifestation of the transient behaviour could therefore be overcome in practice by processing overlapping sequences of data, so the points displayed would always be those that were close to the middle of the sequence. However, this may not be the most desirable way to display the results, if the visualization technique is to provide *visual* as well as probabilistic indications of novelty. It is very clear that the transients at the beginning and end of each sequence are an artefact that has occurred because the visualization

model is not a good one for the data given. If one were to plot the smoothed data with the transients removed by overlapping the sequences, then no such indication that there was anything wrong with the model would be given (and also twice as much processing power would be needed).

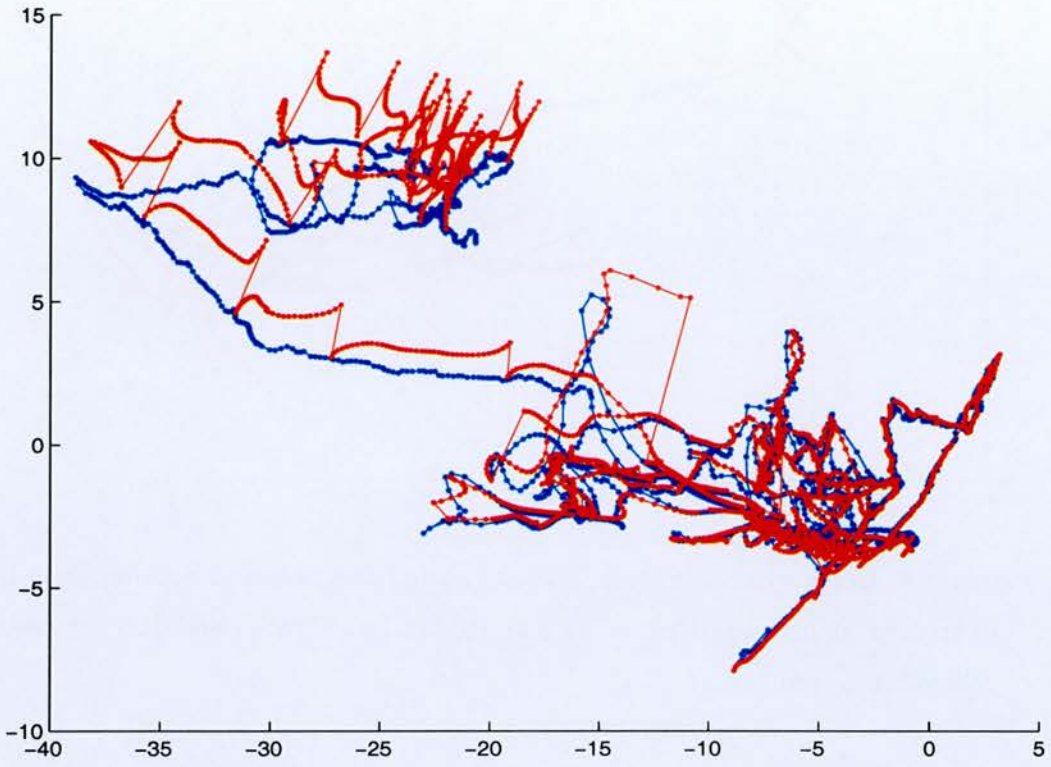


Figure 5.5: Patient 56 data, with online Kalman Filter estimates shown in blue, and sequences of 395 vectors with Kalman Smoother estimates shown in red. Every 10th point is plotted.

As was noted above, the results of Figure 5.4 could not be achieved in practice because the entire data sequence would have to be collected before the blue trace could be computed. A practical compromise is illustrated in Figure 5.5, where the blue trace is the on-line estimates using the Kalman Filter, and the red trace is the result of the Kalman Smoother estimates on the vector sequences of 395 samples. This could be achieved practically, with the blue trace appearing in real time, with the red traces being updated every 395 samples. Note that the end points of each of the Smoother sequences are very close to the Filter estimates. If the Filtering had been done in sequences

of 395, the two would coincide exactly, as the Rauch recursion equations are initialized with the last point set to the Filter estimate.

Figure 5.6 shows the tracking performance of the Kalman Filter along with two indicators of novelty, the "innovation", which is the root mean square of the measurement correction term, and the log probability of each new data point, given the model and the previous observation sequence. It can clearly be seen that from top graph that in the later stages of monitoring, the Kalman filter is no longer able to track the anomalous data accurately (in this case Heart Rate). The transient behaviour that appears in the 2-d principal components plot of Figure 5.5 shows up in this variable.

The detection of "novel" data is also confirmed by the trends in the other two graphs in the diagram; the log probability shows a steady decrease and the innovation term showing a steady increase.

It should be noted that these diagnostic traces would probably be much more marked if a higher order Kalman filter were used, and the state space were extended to the full size of data space, thus precluding the possibility of performing a 2-D visualization. Because the latent space visualization constrains the estimated variables from the Kalman filter equation to lie on a 2 dimensional hyperplane in 4 dimensional space, there will be compromises even during the quasi-stable stage, because the data does not lie in the plane. It is apparent that this effect is present from the form of the probability curve, which is correlated with certain of the variables as they vary in an out-of-plane direction. Particularly notable are the discrete changes in the probability in the early stages, which correlate with the changes in O_2 saturation.

We illustrate in Figure 5.7 the effect of using second order Kalman Filter dynamics in this application. A model was derived, but the resultant visualization did not reveal anything new about the data, and the transient effects at the beginning of each data sequence were much more marked, which produced a confusing and messy plot. The figure plots the first latent space variable plotted as a function of time for a few data sequences in the middle of the novel trajectory. It is apparent from these that the second order model (green trace) is exhibiting decaying oscillatory behaviour at the beginning of the sequence, which is characteristic of an under-damped dynamical system.

Finally, Figure 5.8 compares the Kalman Filter (forward pass only) visu-

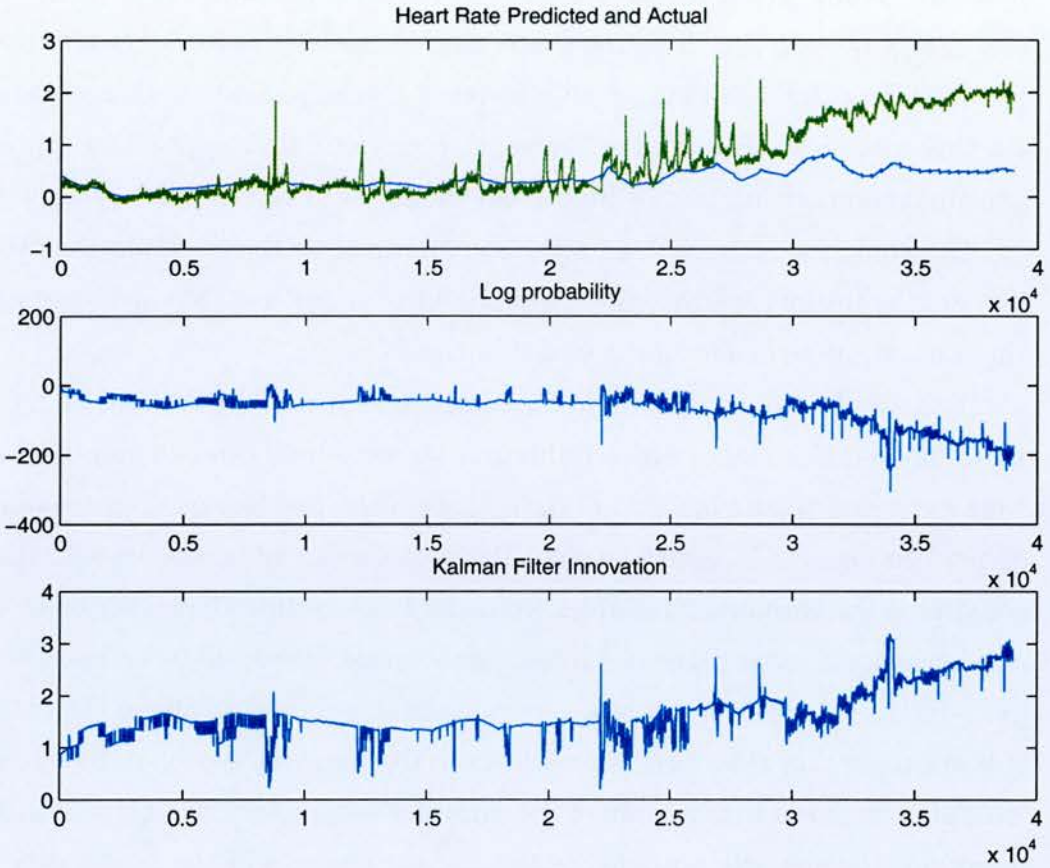


Figure 5.6: Indicators of novel behaviour in PPCA visualizations of "Patient 56". In the top figure, the blue trace shows the model prediction for the variable "Heart rate", and the green shows the actual value. The second graph shows, the log probability of the observation, given the previous observations in the sequence. The third trace shows the Kalman Filter "Innovation" term.

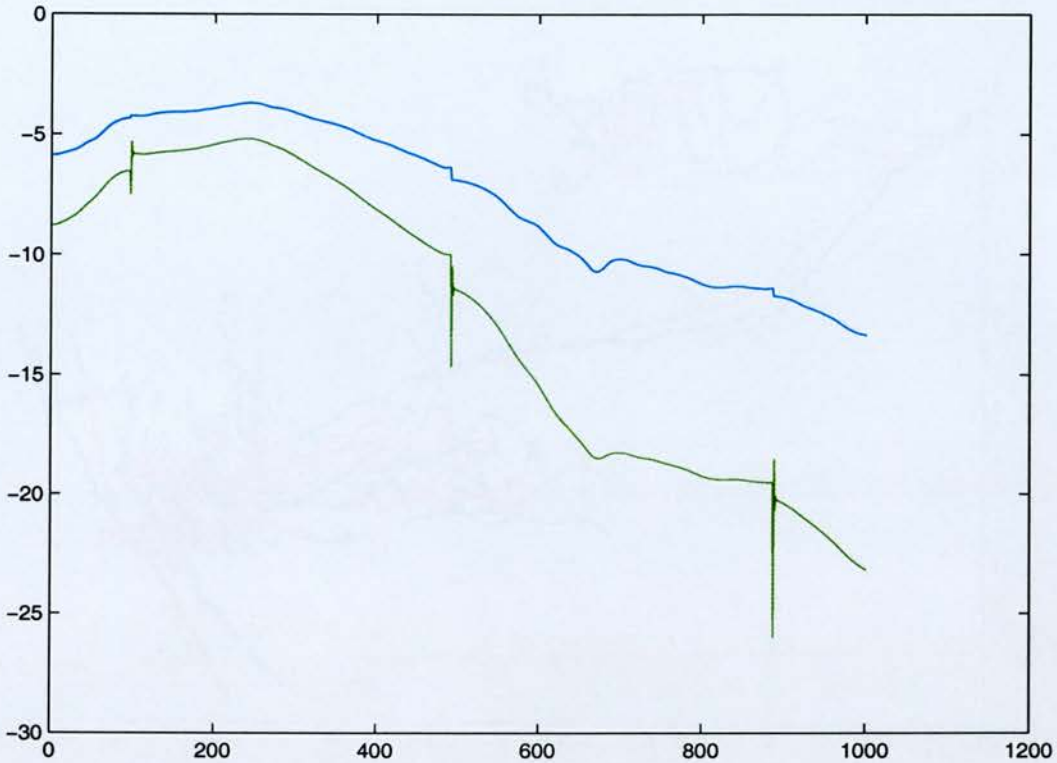


Figure 5.7: Latent variable 1 plotted for Patient 56 during the "novel" phase of monitoring. The blue trace is for first order Kalman Filter dynamics, and the green trace is for second order.

alizations for this data set using first order dynamics (blue) and second order dynamics (red)[3]. The results are much the same as we observed in Chapter 3; the second order system shows a greater degree of smoothing, but in this case, the result is not desirable; valuable information has been thrown away; in particular the incident that occurs when the patient's oxygen mask fell off is less noticeable with the second order system than it is with the first order.

[3]Note that the scaling in this diagram is different from the others presented. This is because the two models produce results that are differently scaled, and so in order to compare the traces side by side, the traces were both normalized to have zero mean and unit variance.

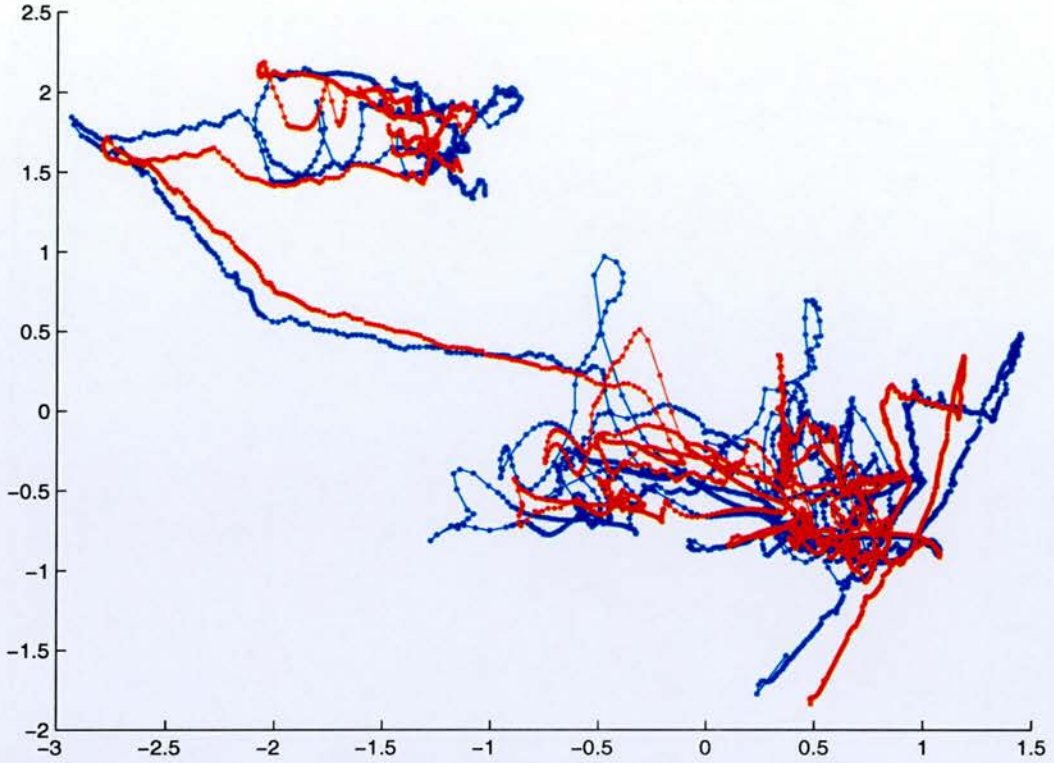


Figure 5.8: Patient 56 data, Kalman Filter estimates for latent variables, with first order dynamics (blue) and second order (red)

5.3.1 PPCATT on Patient 56 without the Oxygen Saturation Data

We noted in the previous section that the O_2 saturation readings for this particular patient exhibited a distribution that did not obey the assumptions made in the PPCATT model, in that it spent a long time saturated at the maximum level, due to the fact that the patient was wearing an Oxygen mask during the monitoring period. In fact, in recent work carried out by the Oxford group[4], it has been debated whether in fact the Oxygen Saturation signal is an appropriate one to monitor, the problem being that if a patient's saturation level drops below 90% they are given an Oxygen mask, that invariably brings up the saturation level to 100%. This means that although the saturation level is high, that such a patient is almost certainly in a worse

[4]L. Tarassenko: Private communication

condition than one who is breathing unassisted with a saturation level of 90%. Saturation levels for a healthy person are in the range $95 - 98\%$.

It is therefore of interest to run a separate experiment, where the O_2 saturation variable was omitted, and the other three variables, (which are more consistent with the assumption of a Gaussian measurement noise model), were used to train the model.

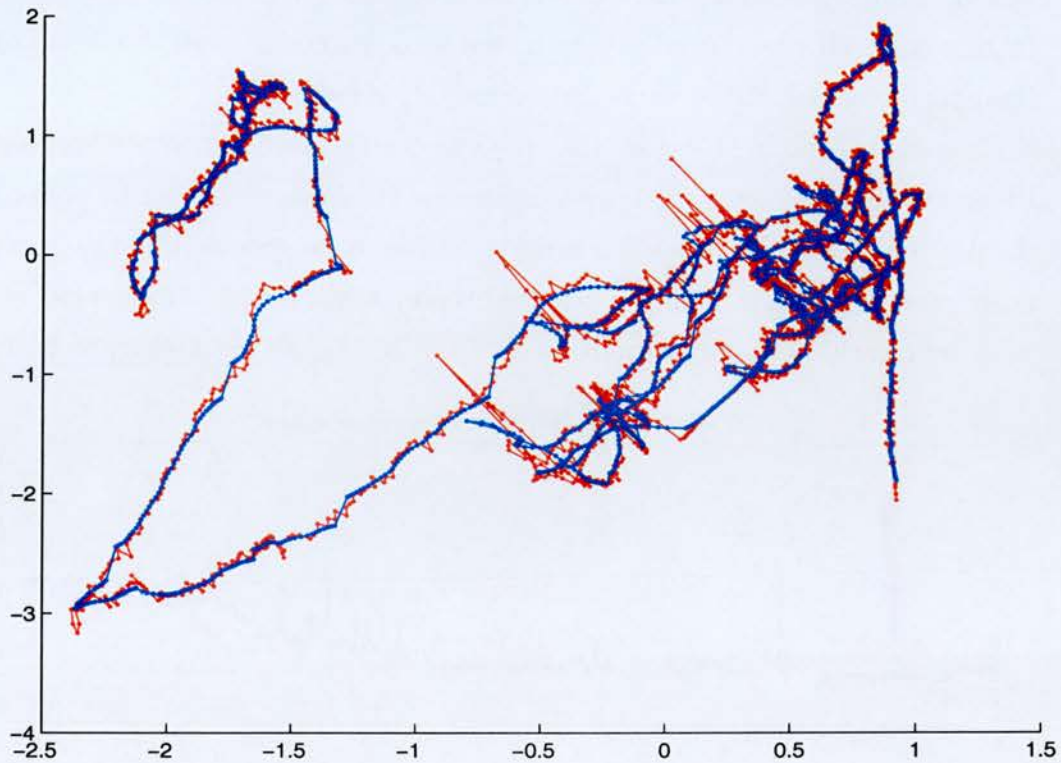


Figure 5.9: Visualization of Patient 56 data, using a PPCATT model that only used the heart-rate, blood-pressure and temperature variables as inputs, omitting the oxygen saturation variable, which was inconsistent with the Gaussian measurement noise assumption inherent in the Kalman Filter. The red trace is the unfiltered data, and the blue trace is the output of the Kalman Smoother.

The results (using a first order Kalman filter for the PPCATT model), are shown in Figure 5.9. It is clear that the display can show the broad trajectory of the patient's progress during the monitoring phase (the early part being displayed at the right hand side of the diagram). The red traces show the unfiltered data, and the blue trace shows the output of the Kalman smoother.

What is immediately apparent is that, while the smoothing effect of the Kalman filter dynamics is still effective, the anomalous behaviour displayed in the latter stages of monitoring, that gave rise to sharp transients in the smoother traces, is no longer present. This is probably due to the fact that the missing O_2 saturation variable varied in a different direction to the other three variables, and hence gave rise to a highly inappropriate linear dynamical model, where the data was well out of the plane of the PCA projection. In this case, all the variables vary in the same direction, and the anomaly (inappropriateness of the visualization model) is less severe.

It is also noted in this case, that the smoothing produced by the Kalman Filter has removed several large excursions in the data, which can be seen on the unsmoothed trace. These turn out to be due to excursions in the patient's heart rate, at around the time when the Oxygen mask fell off. These incidents show up clearly in a plot of the Kalman filter innovation term, (Figure 5.10).

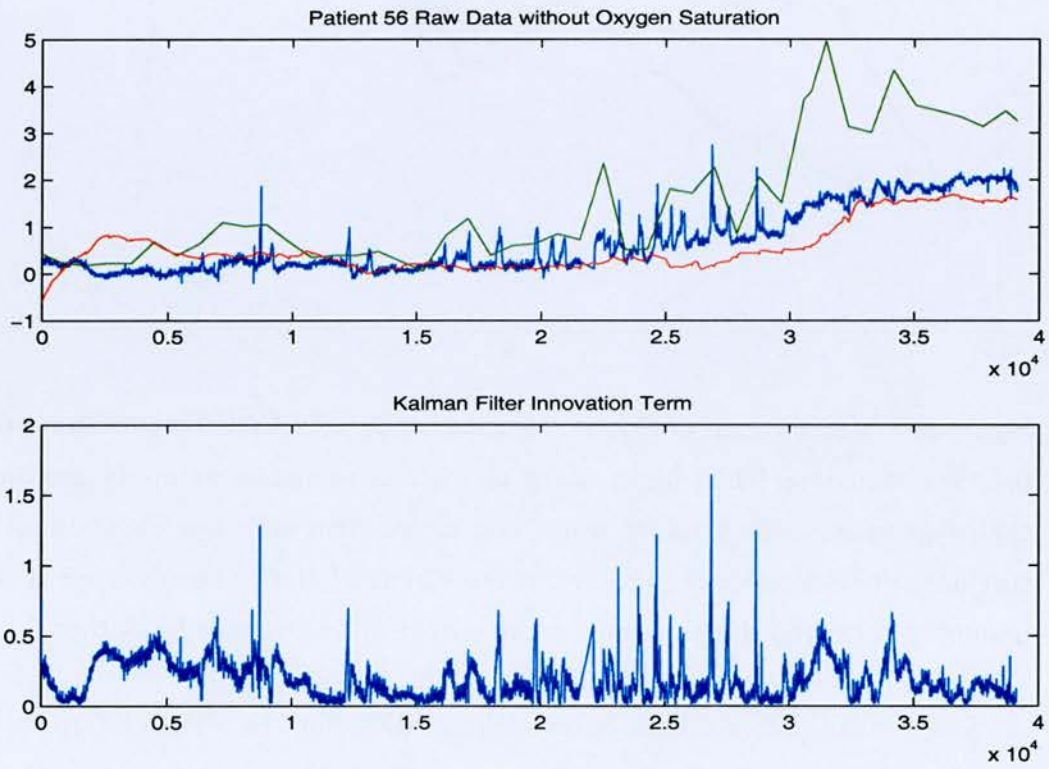


Figure 5.10: Patient 56 Raw data, and Kalman Filter Innovation term, for dataset without Oxygen saturation as an input.

We conclude from this experiment that even without the O_2 saturation

data, the probabilistic model is able to pick up anomalous incidents, which show up as excursions in the innovation term of the Kalman filter (and hence would also show a low probability). It is also noted, however, that the long term worsening trend in the patient's condition, could not be picked up by the probabilistic indicator.

5.4 GTM Through time for individual patients

Six GTM through time models were trained for the individual patients in in the study. The hierarchical model generation scheme outlined in section 4.2.6 was used, where the model was trained in three stages, first on a 10×10 grid, with the initial transition matrix initialized to only allow transitions within a radius of three grid points. After training for 25 iterations, which was sufficient to ensure maximization of the likelihood, the doubling procedure was applied, and a 20×20 model generated, trained again for 25 iterations, and finally, a 40×40 model was produced, again using the doubling procedure. When all the data was used to train the model, the run times were of the order of 3 hours on a 733 MHz Pentium III system.

5.4.1 Progress of monitoring of "Patient 56"

The subject identified as "Patient 56" has already been analyzed using the PPCATT technique, where the learning of the model was restricted to the first 10,000 data points, intending to demonstrate what happens when the PPCATT model encounters anomalous data. In this case, we present results for the GTM Through Time model trained on *all* of the data. The reason for this is that in GTM, the extent of latent space is finite, limited by the grid of delta functions used as the prior distribution. Therefore, when a data point moves out of range of the visualization space, it will not produce an interesting visualization, but will tend to settle on one node at the edge or corner of the map, with a very low likelihood. GTM, GTM Through Time and the SOM all share this behaviour.

Posterior means plots are shown for this subject in latent space in Figure 5.12. In each plot, the trajectory of the entire data set is shown as a blue

trace, and also in red is highlighted a particular section of the trajectory, progressing through the monitoring period, from samples 1–10,000 for the Figure 5.12(a) to 30,001–39,500 for Figure 5.12(f). It can be seen from the plots that there is little evidence of any major change in the first three plots, which wander around in much the same region of the map. However, a definite trend would appear to have set in from around sample 20,000 onwards, and accelerating from 30,000 to the end. This illustrates the rapid decline in the condition of the patient from a quasi-stable state, to an unstable state during the course of the monitoring.

The same data is plotted using the NEUROSCALE mapping in Figure 5.13. This hides some of the detail of the flat latent space plots, but gives a better picture of the magnitude of the changes involved. In the flat plot, the early data points vary from the top to the bottom of latent space, and the final decline in condition does not look more marked than the early variations. Examination of Figure 5.11 shows why this is so.

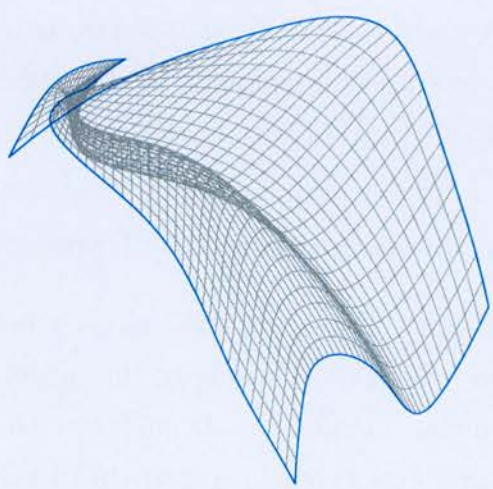


Figure 5.11: NEUROSCALE view of the GTM manifold for "Patient 56"

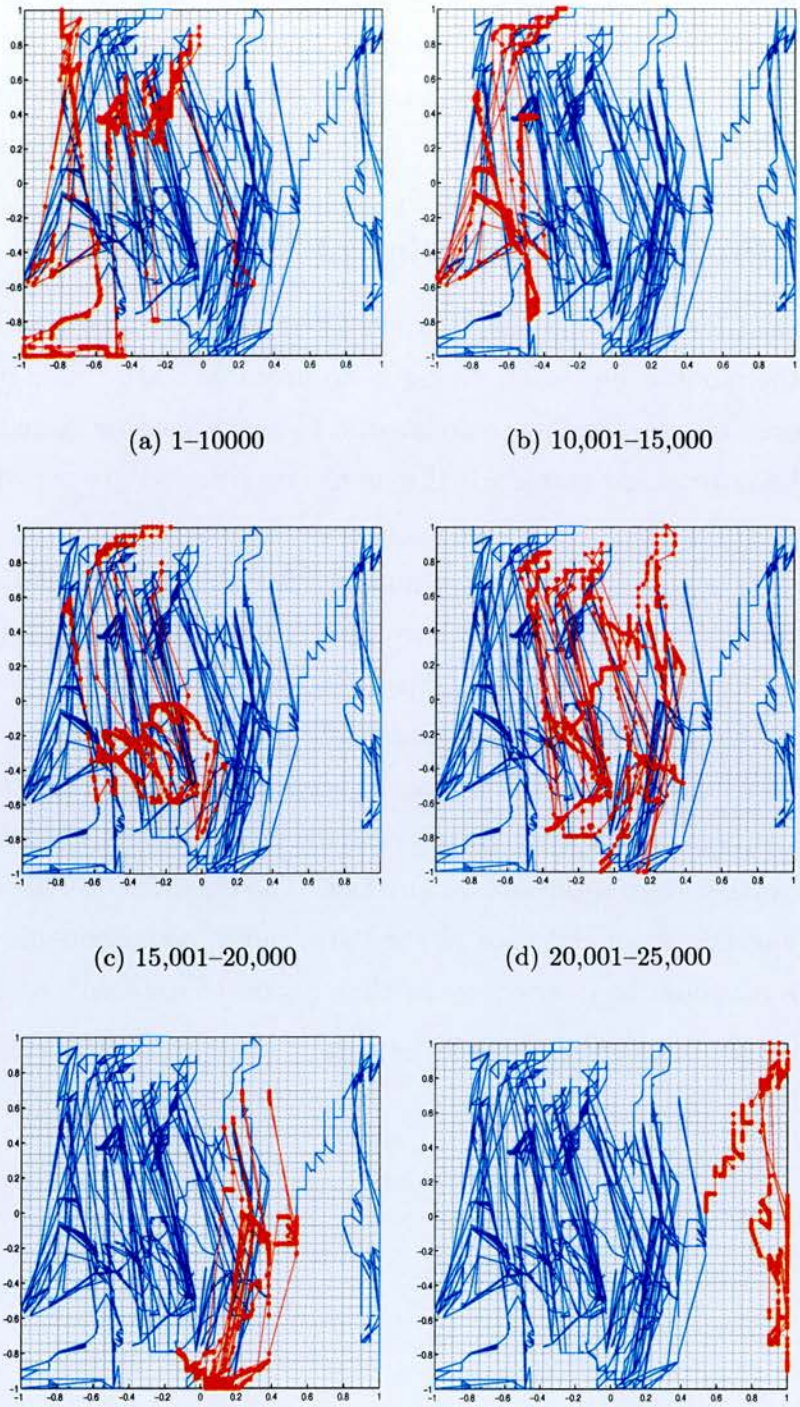
From the figure, note that towards the top left of the diagram, the mesh points are concentrated much more closely together (and NEUROSCALE attempts to preserves data space distances). This would be because a large quantity of the data is concentrated in this area, and hence a probabilistic technique will tend to place a higher density of nodes in this area. Conversely, the mesh points are more spread out towards the bottom right of the plot, be-

cause the data density is lower, and the map thereby becomes more stretched out. The NEUROSCALE view of the data therefore highlights the magnitude of the change at the end of the monitoring period, while de-emphasizing the early variations.

5.4.2 Progress of monitoring of "Patient 37"

The patient identified here as "Patient 37" exhibits a different time history during the monitoring period, being in an unstable state at the beginning of monitoring, and responding to treatment, to reach a stable state towards the end. The normalized traces for the monitoring period are shown in Figure 5.14.

The flat latent space visualizations of the data at different times during the period of monitoring are shown in Figure 5.16. Fig 5.16(c) exhibits two large "excursions" in the highlighted red points. The excursion to the bottom right of the plot is a real one, and corresponds to the large excursion in the "heart rate" variable that can be seen in Figure 5.14 at around record 10,000. It is probable that this is due to an artefact in the measurement process, rather than a physiological effect. The excursion to the bottom left of the diagram is not reflected in the data, but is a consequence of the fact that the manifold has curled up in that region of data space. (See Figure 5.15). The corresponding plots using the NEUROSCALE mapping are shown in Figure 5.17.

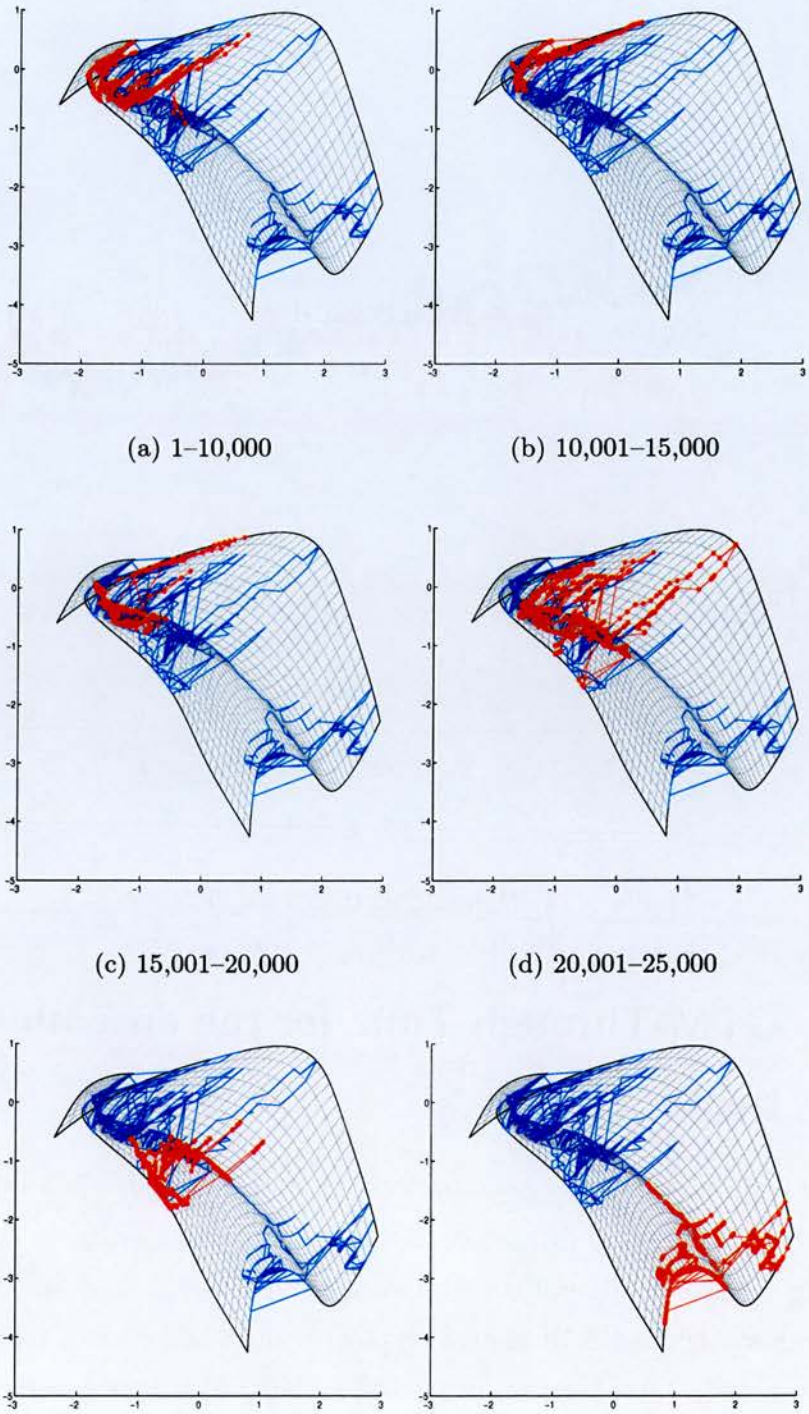


(a) 1–10000 (b) 10,001–15,000

(c) 15,001–20,000 (d) 20,001–25,000

(e) 25,001–30,000 (f) 30,001–39,500

Figure 5.12: Monitoring Progress of "Patient 56", plotting the trajectory of the posterior means in Latent space. A marked trend is apparent setting in at about sample 20,000, and becoming more rapid at around 30,000. In each picture the whole data set is shown in blue, with the indicated section in sample numbers highlighted in red.



(a) 1–10,000 (b) 10,001–15,000

(c) 15,001–20,000 (d) 20,001–25,000

(e) 25,001–30,000 (f) 30,001–39,500

Figure 5.13: Same data as for Figure 5.12 plotted using the NEUROSCALE mapping.

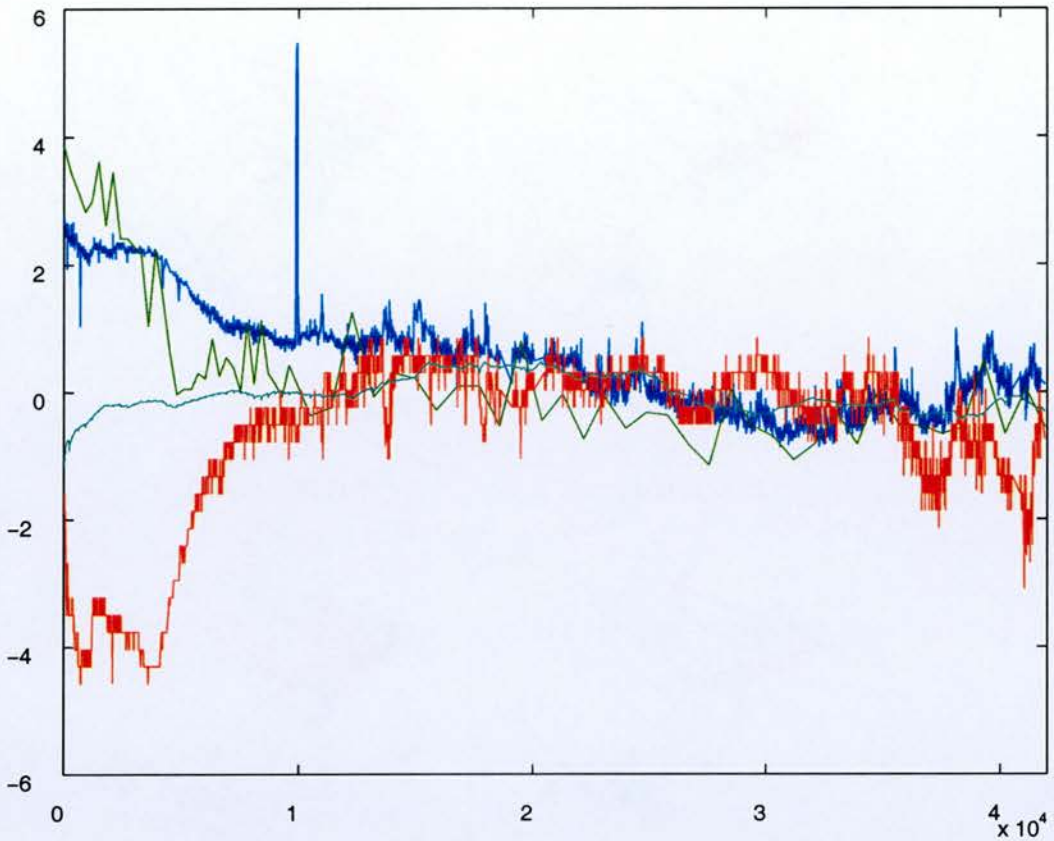


Figure 5.14: Normalized traces for "Patient 37".

5.5 GTM Through Time for the ensemble of patients

Several training runs have been performed on the ensemble of all six patients, using approximately half the data, and training once again in three stages, starting with 10×10, and then building up to 20×20 and finally 40×40.

Figures 5.18 and 5.19 display typical results achievable, using the posterior means in latent space, projected as flat latent space and using NEUROSCALE . In the first two plots shown on each page, the trajectory of the posterior mean is shown, as a blue trace for the patient in question, and as grey points for the remainder of the data set, of which only every 5th point has been plotted.

The plots show very similar characteristics to the results for individual

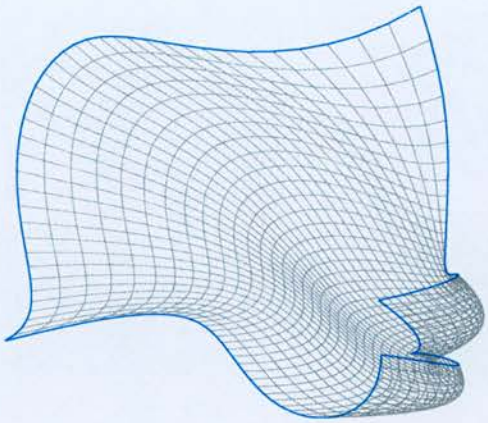
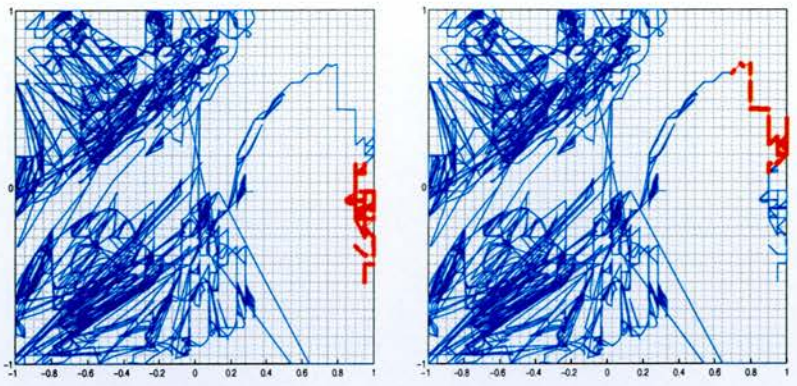


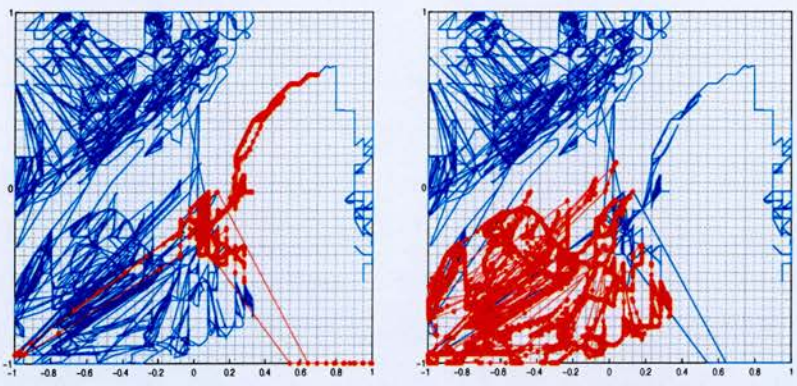
Figure 5.15: NEUROSCALE view of the GTM manifold for "Patient 37"

patients, with the "trajectories" clearly visible. Deduction of structure and scale of the changes in the "quasi-stable" periods of monitoring is less easy from the flat plots, but is more intuitively clear from the NEUROSCALE projections. This is particularly noticeable by comparing Figures 5.19(a) and 5.19(b). The excursion on the flat plot at the approximate coordinates $(-0.7, 0.7)$ in latent space corresponds to the incident where the patient's oxygen mask slipped off. In the flat plot, this feature is roughly the same size as the other features to the right, which are due to the folding of the map. In the NEUROSCALE visualization, the feature stands out as much more significant.

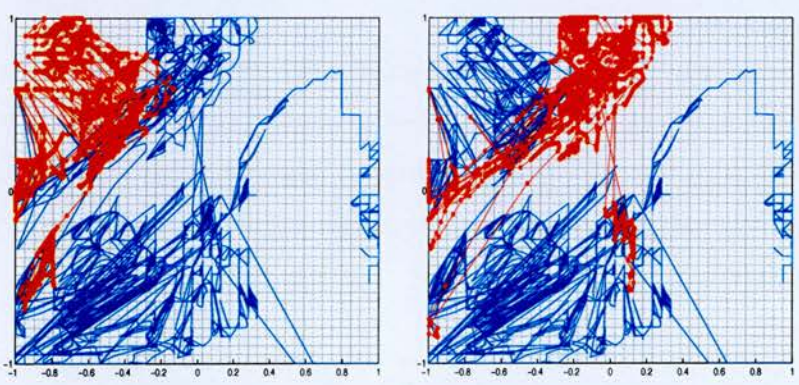
There appear to be two distinct clusters visible especially for "Patient 37", and sudden transitions visible between them. The NEUROSCALE projection lessens the significance of these changes. However this effect is discussed in the next section, using a different training run which displayed the effect more markedly.



(a) 1–3,001 (b) 3,001–5,000

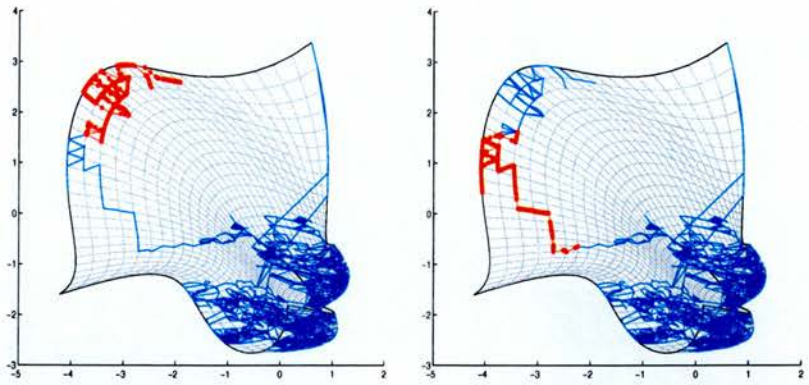


(c) 5,001–1,0000 (d) 10,001–25,000



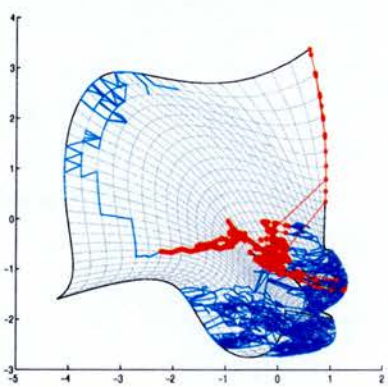
(e) 25,001–35,000 (f) 35,001–41,900

Figure 5.16: Monitoring progress of "Patient 37", plotting the trajectory of the posterior means in Latent space. The patient was in an unstable state at the start of monitoring, which settled down to a stable state after the patient responded to treatment. In each picture the whole data set is shown in blue, with the indicated section in sample numbers highlighted in red.

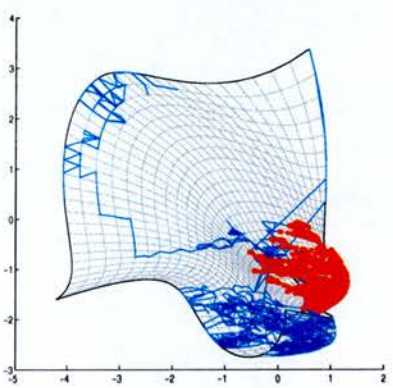


(a) 1–3,001

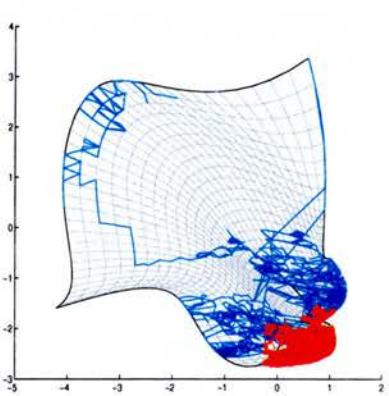
(b) 3,001–5,000



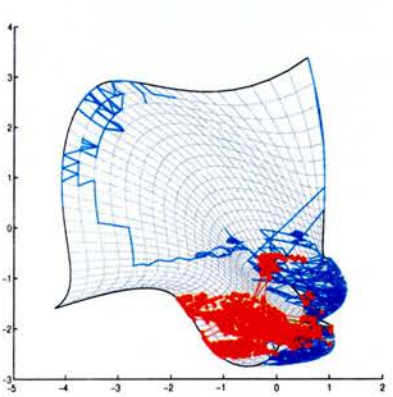
(c) 5,001–10,000



(d) 10,001–25,000

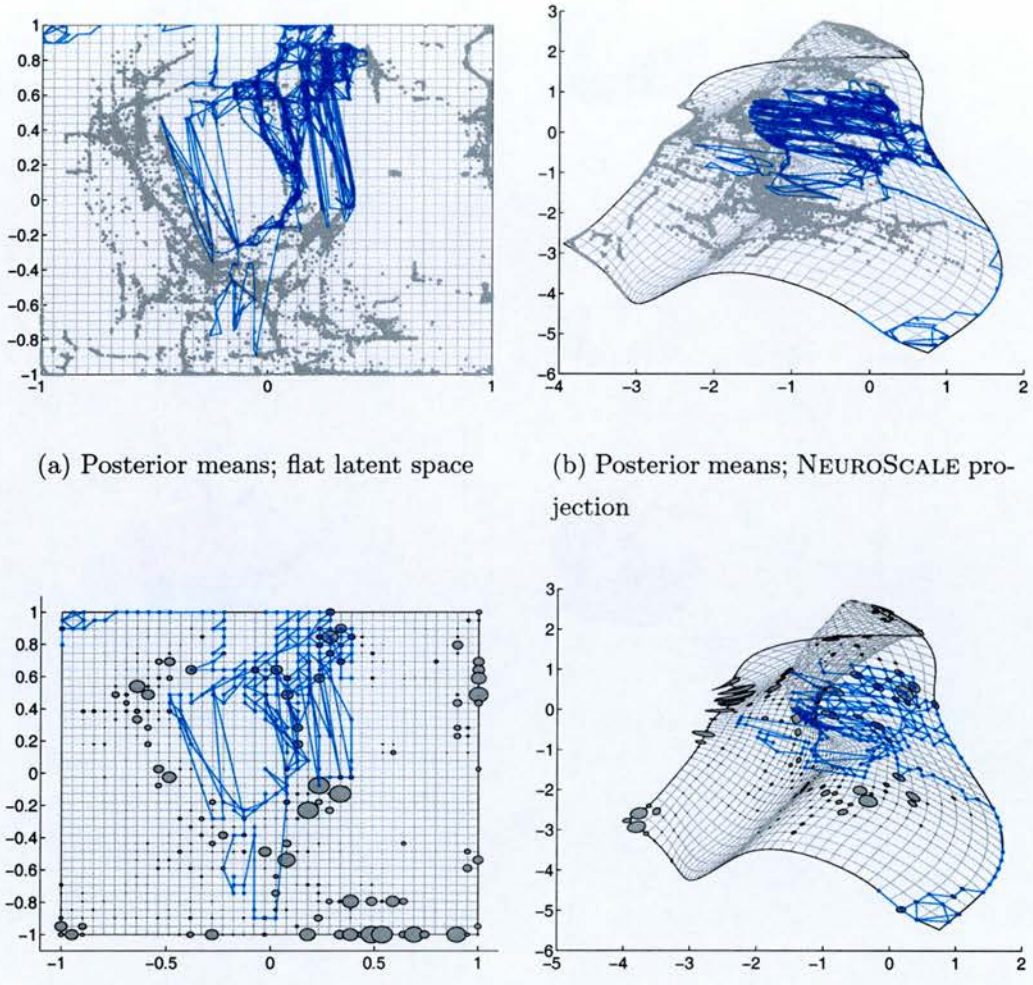


(e) 25,001–35,000



(f) 35,001–41,900

Figure 5.17: Monitoring progress of "Patient 37", plotting the trajectory of the posterior means in Latent space, using the NEUROSCALE projection. The patient was in an unstable state at the start of monitoring, which settled down to a stable state after the patient responded to treatment. In each picture the whole data set is shown in blue, with the indicated section in sample numbers highlighted in red.



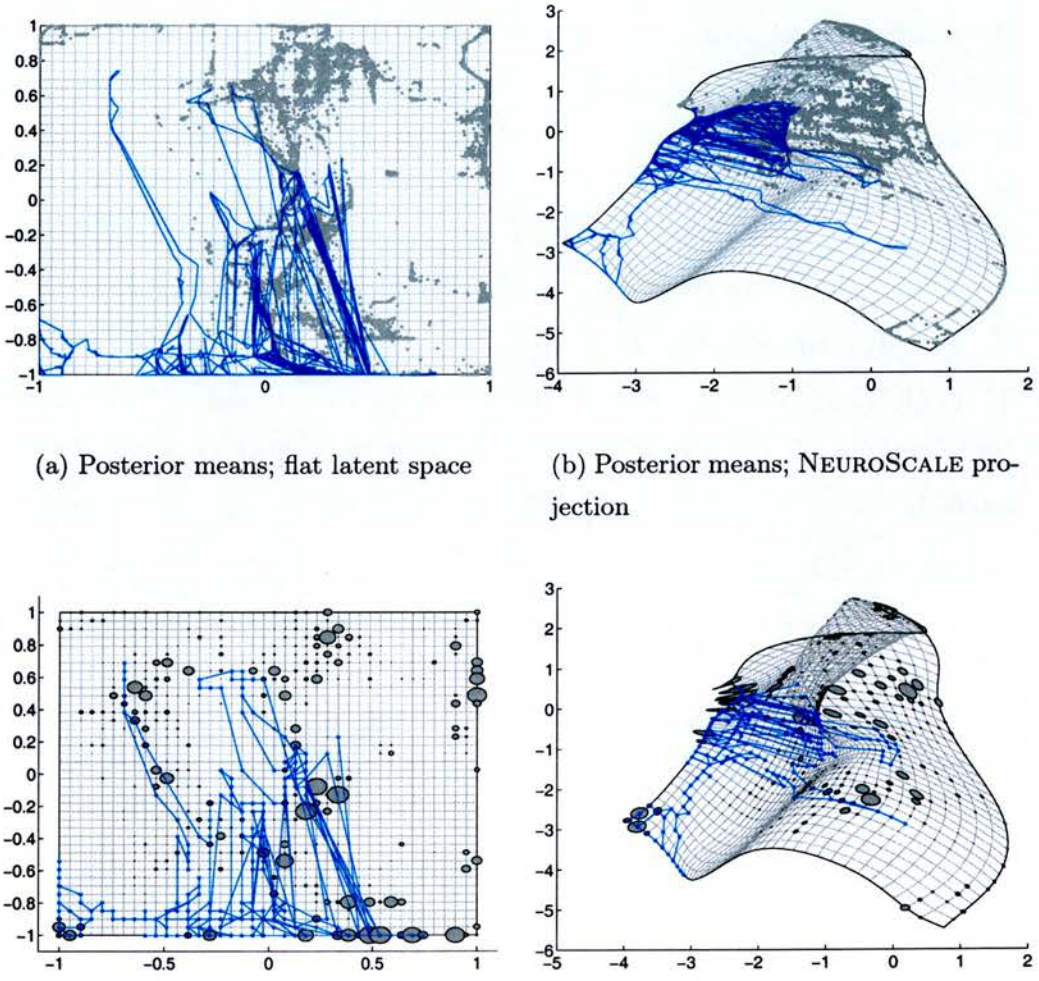
(a) Posterior means; flat latent space

(b) Posterior means; NEUROSCALE projection

(c) Viterbi: Flat latent space

(d) Viterbi: NEUROSCALE projection

Figure 5.18: Patient 37 monitoring seen in the context of the ensemble of patients.



(a) Posterior means; flat latent space

(b) Posterior means; NEUROSCALE projection

(c) Viterbi: Flat latent space

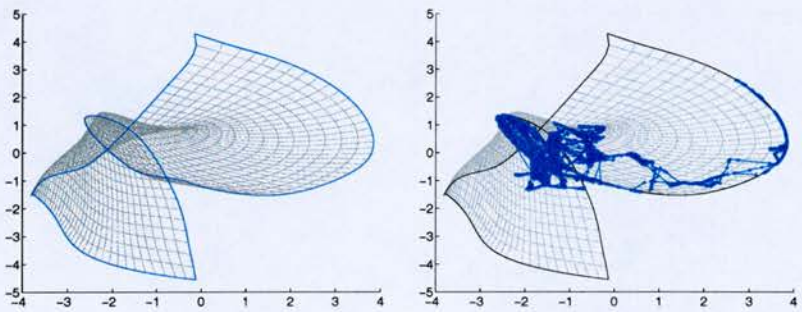
(d) Viterbi: NEUROSCALE projection

Figure 5.19: Patient 56 monitoring seen in the context of the ensemble of patients.

5.5.1 Further aspects of the behaviour of GTM Through Time

A different training run from this data set provides a good illustration of the different degrees of temporal smoothing obtainable using the technique, and also the necessity for having alternate, non-linear views in order to interpret the results. One particular aspect common to GTM, GTM Through Time, and the SOM, is the capability of the derived map to "fold over". This is likely to occur in regions of data space where there is a high density of points, and where the intrinsic dimensionality is greater that two. This behaviour was described in [Kohonen, 1989, pp 156–157], as the formation of "stripes" as the two dimensional map tries to approximate a higher dimensional distribution.

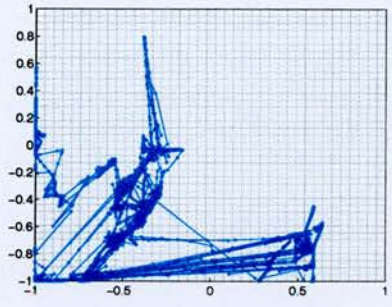
A similar effect is illustrated in Figure 5.20, which is a plot of the GTM Through Time posterior means (using the data for Patient 37) for a fairly "stiff" model (7×7 grid of basis functions with a width of 3 basis function separations).



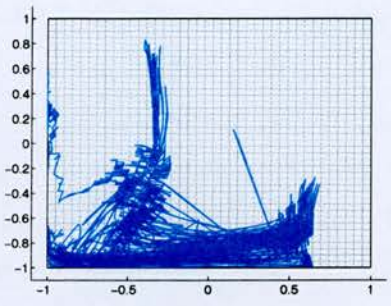
(a) Mesh (b) Mesh+Data

Figure 5.20: Folded over map using NEUROSCALE as a view.

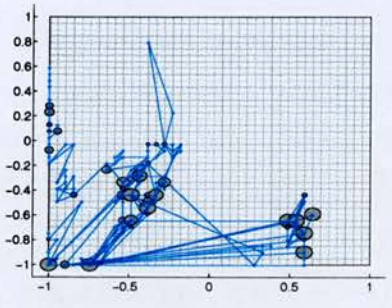
Despite the relatively stiff mapping, it is clear that the map has folded over in a twisted fashion in the region where there is a large amount of data. The NEUROSCALE view of the data has ensured that this cluster of data stays in a localized region of the plot, and has also emphasized the trajectory from unstable to quasi stable condition that occurs at the start of the data set.



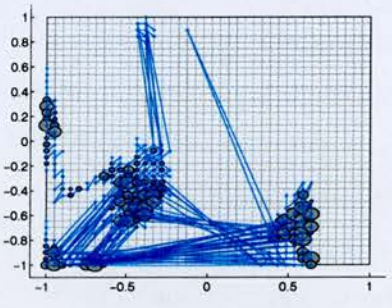
(a) Posterior means



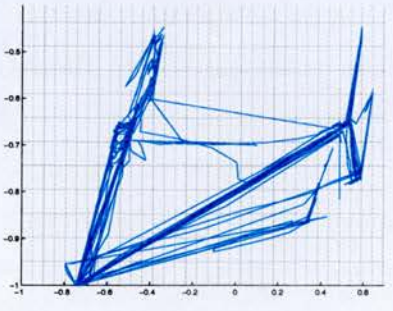
(b) Emission density means



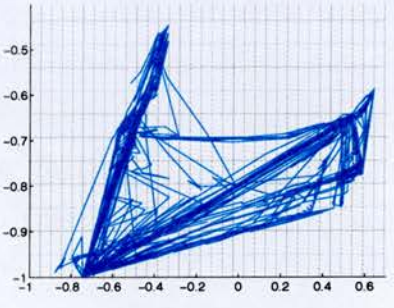
(c) Viterbi



(d) Emission density mode



(e) Posterior means (detail). Sequence length=50



(f) Posterior means (detail). Sequence length=5

Figure 5.21: Different degrees of temporal smoothing available from a GTM through time model, using posterior means (a), the Emission density means (b), the most probable state path (c), and the emission density mode (d). Plot (e) shows the detailed view from (a), and (b) shows the same data, but processed with a vector sequence length of 5 rather than 50, exhibiting a lesser degree of temporal smoothing.

However, Figure 5.21 shows a different, and misleading picture that can be obtained from the "flat" latent space visualization, in that there appear to be two major clusters present late on in the monitoring period. (The trajectory at the beginning appears on the left of the plot). While the model trained solely on Patient 37's data does tend to show a cycle (possibly due to natural diurnal variation), there appear from these plots to be many sharp transitions between the clusters at much greater frequency than once per day. These are like a kind of "jitter" that develops because of the quantization noise on the oxygen saturation signal, which is much higher than for the other signals. However, this "jitter" can be smoothed out to different degrees by the different processing techniques available. It is clear that the straight temporal context has some beneficial effect here, from comparing Figs. 5.21(a), and 5.21(b), where the latter is a plot of the straight emission density means, analogous to the static GTM. Here a great deal more clutter appears on the diagram, and there are many more transitions.

A similar comparison of the difference between temporal and non-temporal processing can be seen in the comparison between Figs. 5.21(c) and 5.21(d), where instead of posterior means, we have plotted the most probable state trajectories, using the Viterbi algorithm in the former, and the Emission density mode (analogous to the SOM "winning node") in the latter. Again, the non-temporal version exhibits more transitions.

Comparison of Figs. 5.21(e) and 5.21(f) shows that some control over the degree of temporal smoothing achieved is possible by varying the vector sequence length. In 5.21(f), the sequence length has been changed to just 5 samples instead of 50, and correspondingly, there are more transitions, and hence more "noise" in the visualization.

There will always be a trade-off involved when data is being smoothed by a process; the greater the degree of smoothing, the more information is likely to be thrown away. In the above example the noise caused by transitions between clusters is an artefact of the fact that the map has a multimodal distribution, and thus is undesirable. However, in other circumstances, it may be that one wishes to reduce the amount of smoothing achieved, to avoid discarding relevant information that is not an artefact of the visualization technique.

5.6 Discussion

In this chapter, we have applied the algorithms developed in previous chapters to a large data set of patient monitoring data. In particular, the records of two patients whose condition changed markedly during the monitoring period, were examined.

In the case of PPCA Through Time, it was demonstrated that the visualization obtained could show a clear trajectory, when the data used to train the model was taken from the early part of the monitoring period, before any marked change. Furthermore, the behaviour of the model for anomalous data was much as expected; with the result being dominated by the slow response of the Kalman Smoother dynamics. This could be used as a good visual indicator of novelty — the model exhibits the artefact of showing transient responses at the start and end of sequences when the prediction of the observed data is poor.

However, the limitations of the model were noted, when the probabilistic nature of the model was used in order to detect novelty. In this case it was found that large variations in probability were often due to the natural variation of the data out-of-plane. This limitation would not be inherent in a full linear dynamical system model, but this would lose the ability to perform visualization.

In the case of GTM Through Time, the objective was purely visualization, rather than detection of anomalous data. We were able to demonstrate the algorithm to good effect, with clear display of the trajectories during periods of change. We compared the results of the new technique with what previous methods could achieve, via different post-processing techniques. In plotting the emission density means, we were able to compare with the kind of output that would be produced by standard GTM (though as noted in the previous chapter, the shape of the manifold produced by GTMTT is somewhat different). Also we compared the kind of output one might achieve from a SOM visualization, by plotting the Emission density modes, and contrasting this with the most likely path of states, as computed via the Viterbi algorithm.

In all cases, it was found that a much tidier, more succinct visualization was achieved via the time-dependent technique. This was due to the effect

of temporal smoothing, which is as we expect from the results obtained from artificial data sets. This is the principal benefit achievable through the time-dependent technique.

The problem of the folding over of the map, as discussed in Chapter 4 is again apparent here. We note once again the trade-off between the conflicting demands of producing a good probabilistic model, and producing a good visualization. It was shown that even the choice of a fairly "stiff" GTM mapping was unable to prevent the folding over from taking place, and that therefore it was necessary to use NEUROSCALE , in order to bring together in the visualization apparent clusters that were mapped to either side of the fold.

The appearance of apparently separate clusters that are really part of the same cluster, on opposite sides of the fold merits some discussion. It might be thought that it would be impossible to jump from one cluster to the other between time points, if there is a locality constraint imposed by the state transition matrix[5]. Such transitions may be thought of as "wormhole" transitions, in that they apparently take a shortcut through a higher dimension in data space, rather than making a smooth trajectory in latent space. The concept is illustrated schematically for a 1-dimensional map, in Figure 5.22.

If a locality constraint on transitions has been imposed by the transition matrix (for example the only transitions allowed are to nearest neighbours), then it will be impossible to get from one side of the "wormhole" to the other, in a single step, as the shortest path through latent space (equivalent to the concept of a "geodesic" in general relativity), would be round the loop. Now consider an unbroken sequence of data points that are in the cluster that spans the "wormhole". If the data are processed as one long sequence, then the latent space point will always stay on one side of the wormhole, because to go via the long route would involve the state path moving out of the region of the cluster, which would have a very low probability. However, if the data are broken up into short sequences that are processed individually, then at the start of each sequence, it is equally likely that the most probable latent space

[5]The example in section 5.5.1 had no locality constraint imposed; the initial 10×10 model was initialized as a dense matrix, so non-local transitions would be possible. However, alternative runs where neighbourhood locality was imposed at the start also showed apparent non-local transitions, and this is what is being discussed here.

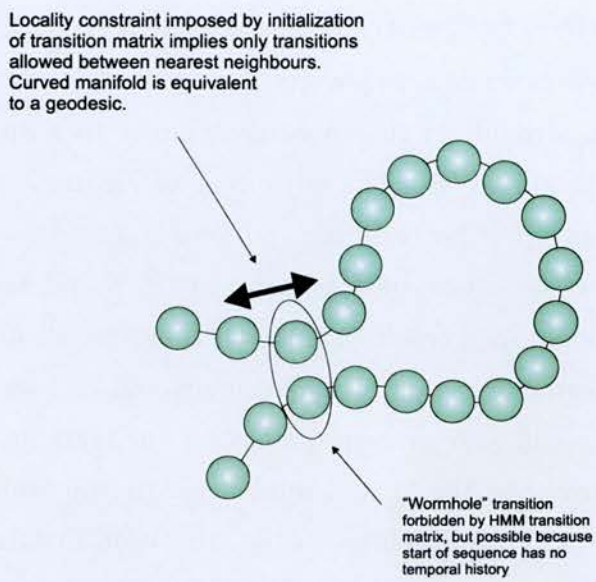


Figure 5.22: "Wormhole transitions" which take a shortcut through 2-d data space instead of the long route through 1-d latent space

point will be on one side or the other. Therefore, if the data is broken into these sequences during processing, and then plotted all as one long sequence, we expect to see these "wormhole transitions" at the sequence boundary. A test was conducted on one of the runs performed on the patient monitoring data set, where the initial transition matrix restricted allowed transitions to a small neighbourhood. The index numbers of all the transitions that exceeded a certain distance were calculated and it was found that these big jumps only occurred at sequence boundaries.

It might be thought that this unsightly artefact in the visualization could be avoided by processing the data in very long sequences. This would indeed decrease the number of wormhole transitions, but the penalty for this is that frequently the path would stay on one side of the fold where from the point of view of maximizing the likelihood, it would be better placed on the other side. Furthermore, it was found experimentally that very long sequences were prone to accumulated rounding errors, which would cause the distribution to collapse. There appear to be two practical solutions to this problem.

The first solution would be to find a method of making the map much stiffer, so that it does not fold over. This might involve modifying the basis functions. The number of non-linear functions could be reduced, or their

width could be increased, and one could employ regularization on the weights attached to these functions, to prevent them having too much influence. This would result in a manifold that was much closer to a linear mapping, with possibly a slight curvature. The advantage of having a visualization which was easier to interpret by avoiding spurious transitions across latent space would, however, be offset by the fact that it would not be such a good probability model. As a result of this, distributions would be more smeared out, and the degree of temporal smoothing would be lessened. Furthermore, such a model would also cause problems for the training algorithm, in that the matrix to invert in the M step would tend to lose rank, and furthermore, the smearing out of probabilities in the transition matrix would mean that the advantages of using sparsity could not be exploited; leading to much higher storage and computational requirements during training.

The second solution is the one that has been used here, namely to use NEUROSCALE to re-map the latent space manifold in order to bring together on the final map the disparate clusters. It was observed that not only did this alleviate the "wormhole transitions" problem, but it also accentuated the big trajectories exhibited by the patients whose condition changed markedly during the monitoring period. This accentuation is not an artefact of the technique; in fact it overcomes an artefact of the flat mapping of latent space, that points where the map is stretched out will be placed closer to the main body of the data than their Euclidean distance in data space would suggest.

In summary, both PPCATT and GTMTT have been applied reasonably successfully to the data set under consideration here. Both models have behaved in the way we expected from the experimentation with toy data sets in previous chapters, but in both methods, drawbacks have also been highlighted, necessitating further research. In the final chapter, we shall consider possible ways in which research into time-dependent visualization techniques might be continued.

Chapter 6

Summary and Future work

6.1 Summary

In this thesis, we have been concerned with developing latent variable methods for visualization of temporal data. Latent, or hidden variables are by definition not directly observable, but their values are to be inferred from observable variables. They are supposed to represent the underlying processes in the data, and influence the observable variables. In using such techniques for data visualization, we are assuming that a good representation of a D-dimensional data set can be achieved with a small number L of latent variables, usually two or three.

In developing latent variable techniques for time dependent data, we have built upon existing work where latent variable models have been used for data sets where the observation vectors are independent and identically distributed (i.i.d.).

The formulation of a probabilistic model for the i.i.d. case involves the selection of a fixed prior distribution in latent space, and a conditional distribution in data space. By application of Bayes' rule, a posterior distribution will be induced in latent space, and the visualization realized by plotting appropriate statistics from this distribution (such as the mean). In extending to the time-dependent case, the assumption is made of a Markov process, where the distribution in latent space at each time step is dependent only on the previous time step. The conditional distribution of the latent variables at the next time step, given the values at the current time step then takes

on the rôle of the static prior in the i.i.d. case. A Prior distribution is also required for the first time step, and this is also learned. As was illustrated (for the case of a Gaussian Prior), in Fig 3.1, the effect of the prior distribution in the inference process is to skew the posterior distribution towards the mean of the prior (in the case of PPCA, towards the origin). It was illustrated that this effect is more marked when the model has less predictive power; that when the model does not predict the data well, the best guess is the prior distribution. This effect, is what is exploited in visualization for time dependent data. The posterior distribution from the previous time step must have a much lower variance than a static prior over the whole of latent space, and hence the skewing effect is much greater. This reflects our prior belief that the observations *ought* to be changing slowly in time. This results in a model that is less sensitive to noise, and hence smooths the trajectory in latent space.

Two particular models have been developed.

6.1.1 Probabilistic PCA Through Time

This model was developed in Chapter 3. It is a simple model that is fast to train, and builds upon the Probabilistic formulation of PCA of [Tipping and Bishop, 1997]. We chose the form for the evolving latent space variables to be a linear dynamical system with a Gaussian noise vector, leading to a Gaussian form for the conditional distribution $p(\mathbf{x}_{n+1}|\mathbf{x}_n)$. This gave an E-step that uses the Kalman Smoother algorithm to compute the expected values of the latent variables, and an M-step which involves the maximization of sums of quadratic terms.

It was demonstrated that the algorithm could be extended to higher order linear dynamics while retaining the constraint of the latent variables lying in the plane of visualization space.

We demonstrated the use of PPCA through time on toy data, and also on patient monitoring data. The behaviour of the model was as expected, in that it showed the capability to form smooth trajectories in latent space, governed by the linear dynamics of the Kalman Smoother.

It has also been observed (for the Patient Monitoring Data), that the technique reveals the presence of "novel" data (the decline in stability of the condition of one of the patients), when training has taken place in the absence of the novel data. This indication of novelty manifests itself in two ways, namely by a decrease in likelihood, and also in the appearance of "artifacts" due to the transient behaviour of a linear dynamical system when there are large reconstruction errors in the PCA mapping. These artifacts are a manifestation of the behaviour described in the previous section; when the model has poor explanatory power for the data, the prior tends to dominate.

A key limitation in the model is inherent in the requirement to produce a visualization, which conflicts with the requirement to produce a probabilistic model capable of assigning meaningful probabilities. If visualization is required, then the latent space dimension is limited to two, or at most three, and the mapping is linear. Therefore, it is likely that some of the data, though quite normal, will lie out-of-plane, and therefore be assigned a low probability. This means that low probability will not always imply abnormal data. If visualization were not a requirement, then the dimension of latent space could be increased to produce a better probabilistic model.

A second limitation of such a model is that the dynamical model is assumed to be the same at any point in latent space. This means that such a system would be a poor model, for example for the Lorenz Attractor example, where there are "eddies" that circulate in opposite directions at different parts of latent space.

6.1.2 GTM through Time

The GTM through Time model of Chapter 4 has a non-linear manifold in data space with a multinomial prior distribution, that is discretized on a regular grid. The mapping from latent space to data space is accomplished by a Radial Basis Function network. At each new measurement, the prior distribution in latent space is produced by taking the posterior distribution from the previous measurement, and multiplying by a transition probability matrix. This contrasts with the i.i.d. case for GTM, where the prior is a uniform distribution over the grid points.

The model has been developed and demonstrated on synthetic and real data. Potential problems due to the large number of adjustable parameters (in the transition matrix) have been discussed, and shown not to be a problem, either because most of them decay to negligible values (because in temporally correlated data, only a small range of transitions come from a single point), or because in a highly constrained system such as GTM, the parameters connected to adjacent grid points will be smoothed out.

A key theme in the implementation of GTM Through Time has been that of sparse representations of the various matrices involved in the model, and in the computation. In addition, we showed that GTM itself, by exploiting the inherently sparse nature of the matrix of basis functions, can in principle be scaled up to very large sizes. This was originally considered not feasible for GTM because the computational complexity of the M step, where a set of linear equations must be solved, is of order M^3, where M is the number of basis functions.

We demonstrated how sparsity can be exploited further in GTM through time, in the treatment of the transition matrix. It is possible either to initialize the transition matrix to a sparse form, by incorporating prior knowledge of the problem (e.g. in only allowing localized transitions in latent space, because data changes slowly through time). As an alternative, if the transition structure of the data is unknown (or if some rapid non-local transitions are known to occur), one can initialize the transition matrix densely, and let the sparsity pattern develop. The process can be accelerated by applying a threshold below which a probability can be treated as zero, rather than allowing this to occur "naturally" by relying on the numerical (im)-precision of the machine. We demonstrated clear instances of specialized sparsity patterns being learned in the "Switching state robot" example.

In terms of the kind of visualization achievable, we showed, that, in line with our expectations, the model was able to filter out uncorrelated noise in the visualization, while still retaining the ability to learn temporally correlated trends. We also demonstrated empirically, and gave theoretical justification for the fact that the temporal version of the algorithm gives a qualitatively different mapping from the static version. Specifically, the map

produced was more sharply curved, reflecting a better density model, and representation of the data.

However, this last observation leads to one of the problems inherent in this kind of visualization technique. Because the standard way of visualizing data using non-linear techniques is to "flatten out" the manifold that is learnt in latent space, this tends to lead to a less intuitive way of visualizing the data. We noted this in the patient monitoring example, where the anomalous trajectories had the same magnitude as smaller scale variations within the body of the data. A related problem occurs when the manifold "folds over" in data space, resulting in a bi-modal posterior distribution. In a "flat" visualization, such a distribution will either result in anomalous points on the plot, representing the mean of a bi-modal distribution, or many rapid transitions will occur between the two peaks of the distribution, which do not in fact correspond to very significant changes.

One solution to this problem of scaling has already been proposed in the use of "Magnification Factors" [Bishop et al., 1997b], which allow the flat map to be colour-coded according to the degree of stretch. In this study, we introduced an alternative technique, based around the topology preserving NEUROSCALE mapping [Tipping and Lowe, 1998], trained using the grid points on the GTM manifold. This provides first a useful technique for examining the manifold itself. In order to interpret the clusters we see on a flat GTM plot, it is helpful to know how twisted, or stretched out, the manifold is. NeuroScale, with its distance preserving properties, appears to be a natural choice. Furthermore, NeuroScale can then be used to visualize the data itself, and display its position on the grid in Latent space. We demonstrated in Chapter 5 in the case of the Patient Monitoring data, that this restored the scaling to the features in the data set. We have thus combined the distance preserving properties of NeuroScale with the probabilistic framework embodied in GTM and GTM through time.

This technique clearly is also applicable to the standard GTM and also to the SOM.

6.2 Suggestions for future research

6.2.1 Mixtures of Models

This would be potentially a very fruitful area of research. To extend PPCA through time to a mixture of PPCA models through time would overcome the following limitations of the current model:

- In having a mixture of models we overcome the limitation of having to fit a two-dimensional plane through all the data, leading to low values of likelihood for normal data that lies out of the plane. This extension has already been achieved for the i.i.d. case [Tipping and Bishop, 1999], and it has been possible to reveal more structure within a dataset than with a single PPCA model.

- The separate mixtures would each have their own linear dynamical models, overcoming the limitation noted above of having one set of dynamical equations for the whole of latent space.

The proposed model would be a hybrid between a Hidden Markov Model (modelling the transitions between mixture components), and the Kalman Filter, (modelling the linear dynamics in each state). This type of model is considered in [Ghahramani and Hinton, 1996b]. The corresponding graphical model is shown in Figure 6.1.

The symbols $\mathbf{x}_i^{(m)}$ refer to the mth mixture component, whose internal dynamics are governed by the Kalman-Smoother recursions. The symbols S_i refer to the ith state of the system, where switching transitions between states are governed by the state transition matrix. Since each state space model learns its own set of dynamical equations (corresponding to $\{\mathbf{A}, \mathbf{b}, \mathbf{W}\}$ in PPCA through time), it would enable different dynamics to be simulated in different regions of data space. This type of model is considered in [Ghahramani and Hinton, 1996b]. The principal difficulty is that the E-step for such a model is no longer tractable. The convergent connections of arrows on the observable variables for each of the state space models and the switching state variable implies that they are all coupled via the switching state variable. Hence the number of possible paths through the graph increases

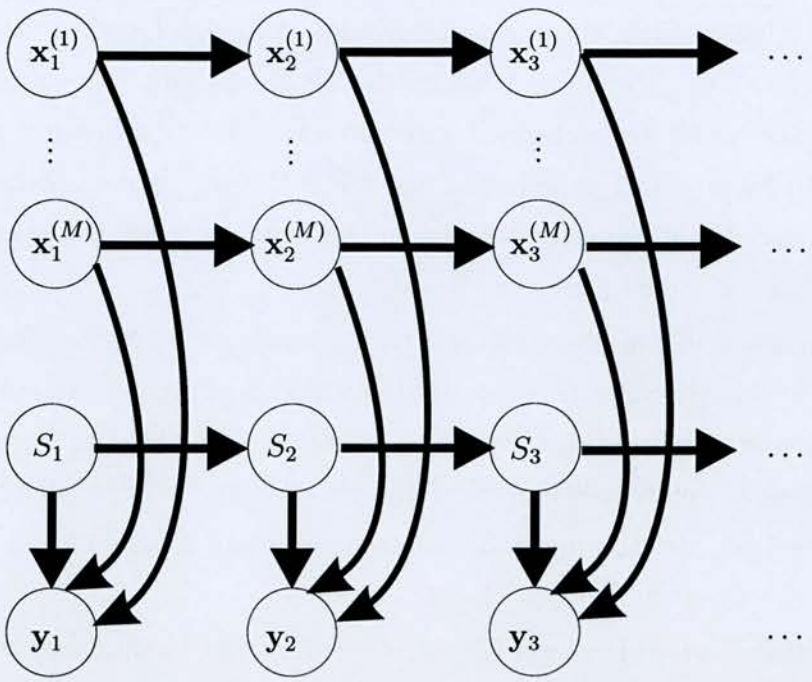


Figure 6.1: Probabilistic Graphical Model for Mixtures of PPCA Through Time

exponentially with the sequence length. If there are M state space models, at T steps in the sequence, then M^T paths must be evaluated, which would restrict the model to very short vector sequences and small numbers of components. Ghahramani and Hinton overcome this problem by using a variational approach, where a more tractable distribution is used to approximate the actual distribution over the hidden variables. In this case, the vertical arrows in the graph are dropped, and the distribution is treated as a set of uncoupled state space models. This then requires an iterative E step, which minimizes the Kullback-Liebler divergence between the approximating distribution and the actual distribution. This process is complex, and also prone to local optima. The risk of falling into such optima can be reduced by a deterministic annealing process on the cost function.

Therefore, while the mixture of PPCA through time would be an attactive model, it turns out that the computation of the E step is far more complex than in the straight PPCA through time. However, the model has many less parameters than GTM through time.

Visualization of the results from a piecewise linear model involves the display of several plots simultaneously, one for each linear component. An

interactive hierarchical visualization technique was developed in [Bishop and Tipping, 1996], for the iid case where the division of analysers up into sub-components could be performed interactively. Such techniques could be applied to hierarchical time dependent visualization, though some form of feedback should be given to show when the state switched from one plot to another (e.g. a dotted line).

Extension of GTM through time to a mixture of GTM manifolds might be simpler conceptually; separate RBF's would perform the mappings from the particular patch of latent space to data space. In the simplest case, one large transition matrix could cover the transitions between all the states, but in practice, one would reduce the number of parameters by the grouping methods discussed in Section 4.2.5.1.

It is likely that this would be an interesting area to pursue, because the ability to have multiple non-linear manifolds spread across data space would probably result in a reduction of the amount of curvature in each individual map, because it would not have to make a good fit to all of data space. Thus the maps should have less tendency to fold over and produce convoluted maps where the interpretation is difficult in the flat latent space plot.

6.2.2 Non-linear state space methods

An alternative to having the discretized latent space of GTM, combined with the smooth output mapping of an RBF function, is to have a continuous latent space, where the latent variables evolve according to *non-linear* dymamics. This type of model has been proposed in [Roweis and Ghahramani, 2000], where the E step involves using the Extended Kalman Filter algorithm, which uses a local linearization of the non-linear model at each step, and where the M-step is made tractable by defining the non-linearity as a sum of Gaussian Basis Functions. Ghahramani and Roweis suggest that for low-dimensional latent space, the basis functions can be placed on a uniform grid, resulting in a model that uses very much the same formulation for the latent space to latent space time-dependency as for the latent-to-output space mapping that is used in GTM and GTM Through Time. Because the RBF is a universal function approximator, any embedded flow field can be

learnt by such a model. This is analogous to GTM through time, which was demonstrated in Chapter 4 to be able to learn the non-linear dynamics of the Lorenz Attractor (shown in Fig. 4.12). It may be the case that a more parsimonious representation of the dynamics is possible using the sum of basis functions, rather than the use of the discretized state grid in GTM through time.

Another possible model using this formalism would be to have linear dynamics in latent space combined with a non-linear output model as used in GTM. This would be another way of overcoming the problems encountered where PPCA through time would give a low likelihood for out-of-plane variations. Such a model would simply be a special case of the general formalism presented in [Roweis and Ghahramani, 2000].

6.2.3 On-line methods

All the algorithms presented in the thesis have been batch based methods, where all the data sequences have been collected prior to training. However, there would be considerable advantage in having on-line versions of the algorithms, for monitoring applications; where the data is collected in real time; a monitoring trace displayed, and the data thrown away.

Adaptation of the dynamical systems based models for online learning is discussed in [Roweis and Ghahramani, 2000], where the key idea is that the cost minimized in the M step is a sum of quadratic terms, whose solution therefore involves a solution of a set of linear equations. These are thus amenable to an on-line recursive least squares method of solution. In such a system, the expectations are computed using the Kalman *Filter* (or Extended Kalman Filter in the non-linear case), and not the Smoother, because we do not have the entire sequence of observations available.

Likewise, an extensive literature exists for online learning of Hidden Markov Model parameters, e.g. [Ford and Moore, 1998] and [Stiller and Radons, 1999]. A typical scheme is to operate on a single observation sequence at a time, and perform a single forwards-backwards pass, initialized with the specified prior. This gives a new estimate of the expected transition statistics, and this is then used to update the transition matrix by taking a weighted

sum of this quantity and the old expected transition statistics. The size of the weighting coefficient (between 0 and 1) determines how quickly the past is "forgotten".

6.3 Conclusion

In conclusion, we note that the field of visualization techniques of data through time appears to be on a solid theoretical foundation, based around latent space models.

Principally in this thesis, we have presented a simple model, based on linear dynamical systems, evolving in a linear output space, in the manner of PCA, and a more complex model, that can in principle emulate any non-linear dynamical time evolution of data.

In the preceding section, we have placed these two models for visualization in the context of a more general framework, within which it appears that a whole class of new models could be derived. There seem to exist, therefore, many possible avenues of research in this field. It seems appropriate, therefore, to end this thesis with a quotation from T.S. Eliot's famous set of meditations on the nature of time and timelessness, "Four Quartets":

> *In my end is my beginning.*
>
> From "East Coker"

Appendix A

Data sets used in this thesis

A.1 Lorenz Strange Attractor with added noise

The Lorenz Attractor is a well-known example of chaotic non-linear dynamics. It is defined as the time evolution of the following set of coupled non-linear differential equations, which are a simple model of medium scale atmospheric convection.

$$\dot{x} = -\beta x + yz \tag{A.1}$$

$$\dot{y} = \sigma(z - y) \tag{A.2}$$

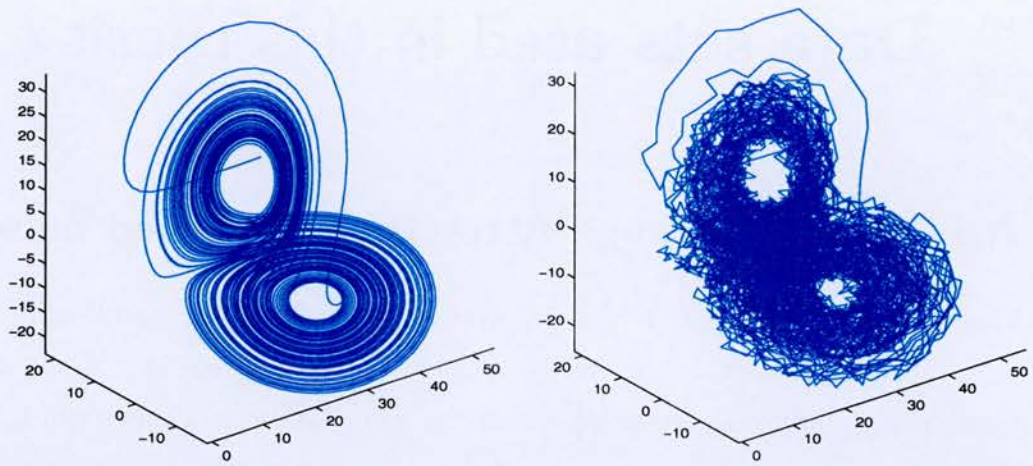
$$\dot{z} = xy + \rho y - z \tag{A.3}$$

The equations exhibit chaotic behaviour in that the long term values of the variables are highly sensitive to initial conditions, and the smallest change to the starting conditions will lead to a rapid divergence of the two sets of solutions. This behaviour led to the term "butterfly effect", coined by Lorenz in a talk to the American Association for the Advancement of Science in 1972, with the title:

> Predictability: Does the Flap of a Butterflys Wings in Brazil set off a Tornado in Texas?

Although the time evolution of such a system is not in general predictable (because it will be adversely affected by rounding errors), the general dynamical behaviour is well-understood; the variables evolve as a set of slowly

growing oscillations that periodically flip between different "modes", where the centre of oscillation changes. This is illustrated in Fig. A.1(a), where 10,000 time steps of 0.01 have been computed using a standard ODE solution package, with $\sigma = 10, \beta = 28, \rho = 8/3$, and the initial conditions chosen randomly.



(a) Clean (b) Noisy

Figure A.1: Lorenz Attractor data with and without added noise.

This behaviour is interesting from the point of view of the methods developed in this thesis, because it represents two types of temporal evolution; relatively smooth changes most of the time, with occasional bursts of rapid changes produced by the flipping between states. The system is also of interest in that it is inherently suitable for two dimensional visualization, in that the points in (x, y, z) occupied form a lower dimensional manifold, known to be of a "fractal dimension".

In order to illustrate the smoothing effect of taking temporal context into account, we have added noise to each point after it has been calculated by the ODE solver, drawn from a spherically symmetric Gaussian distribution, with variance of unity. Note that this does not effect the simulation of the dynamics of the system, as it has been added after computing the dynamics.

It should therefore be regarded as "measurement noise". The resultant trace is illustrated in Fig. A.1(b).

A.2 Switching state robot

The problem of the kinematics of a two-link robot manipulator has been studied elsewhere in the machine learning literature ([Bishop, 1994]). The robot manipulator is represented schematically in Fig. A.2.

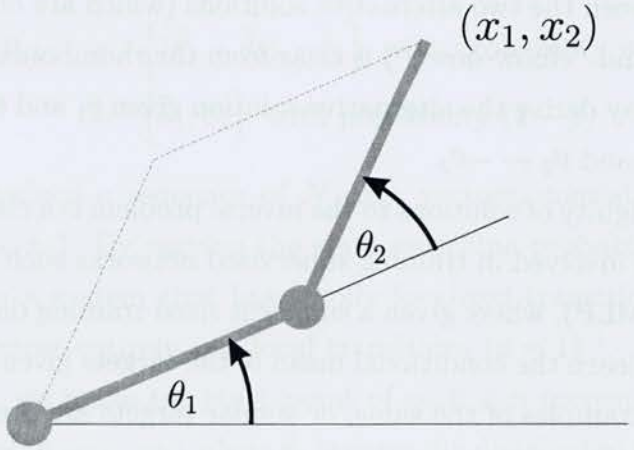


Figure A.2: Schematic of robot manipulator with alternative solutions, corresponding to "elbow up" and "elbow down" configurations.

The figure shows a simple schematic of a two-jointed robotic manipulator. A principal problem in robotics is the solution of the so-called *Inverse Kinematics* problem, where it is required to compute the required joint angles (θ_1, θ_2), in order to position the end-effector of the robot to a desired location in Cartesian coordinates (x_1, x_2). The corresponding *Forward* problem has a unique solution with respect to the joint angles:

$$x_1 = \cos(\theta_1) + \cos(\theta_1 + \theta_2) \quad \text{(A.4)}$$

$$x_2 = \sin(\theta_1) + \sin(\theta_1 + \theta_2) \quad \text{(A.5)}$$

However, the solution of the inverse problem can be seen to have alternate solutions:

inputs derived from the flight recorder charts were selected. Classification was performed manually. For the purpose of this study, we have removed one of the classes, corresponding to when the helicopter was on the ground prior to take off, as it was well separated from the other classes, and thus tended to define the principal axis, thus hiding the details of visualizations during the time when the helicopter was actually in the air. In removing one of the classes, it was also found that one of the inputs ("Ground Effect") was now redundant, having a constant value throughout the data set. This was then also removed, leaving a data set of 914 points. The normalized data is plotted in Fig. A.3.

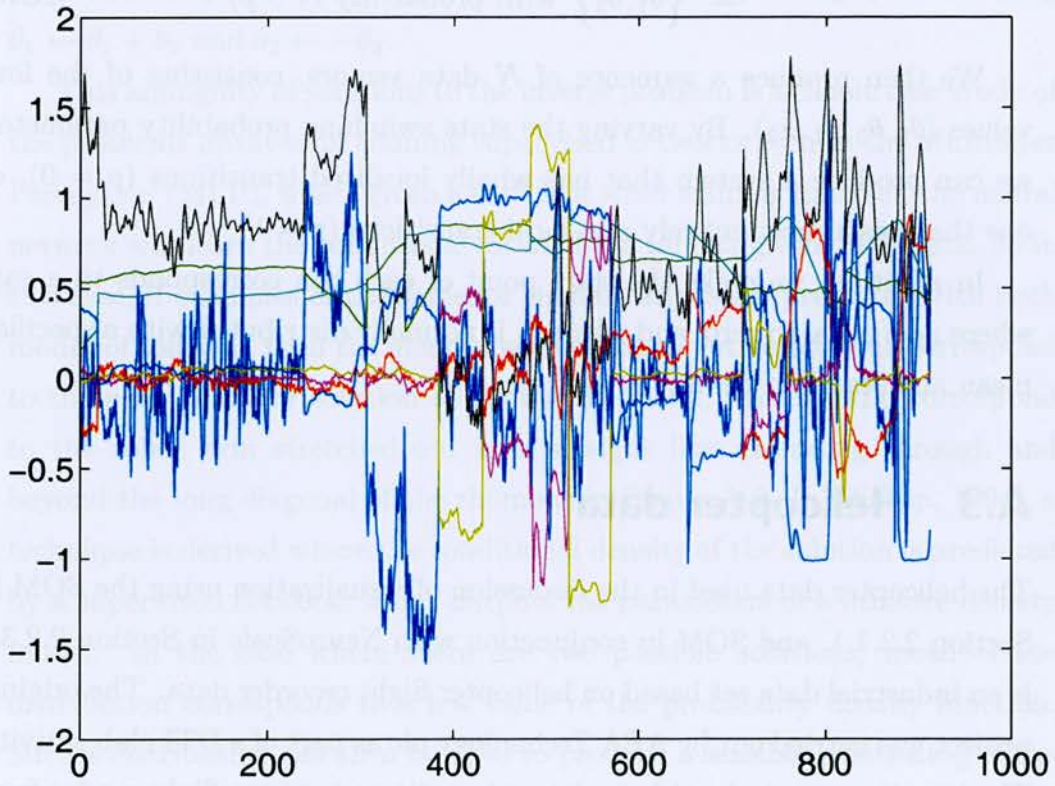


Figure A.3: Plot of normalized helicopter input data.

Appendix B

The Viterbi Algorithm

The Viterbi Algorithm [Viterbi, 1967], [Forney, 1973], is the standard technique for computing the most probable sequence of states of a Hidden Markov Model (HMM) [Rabiner, 1989], given an observation sequence, and its emission probabilities, and the prior and transition matrix of the HMM.

The Hidden Markov Model is defined by the parameter set $\lambda = \{\mathbf{A}, \boldsymbol{\pi}\}$, where $\mathbf{A}$ is a $K \times K$ matrix where a_{ij} is the probability of the state transition $S_i \rightarrow S_j$ occurring. It is called the *Transition Matrix.* $\boldsymbol{\pi}$ is a vector of length K giving the prior state probability distribution. As input to the algorithm, we have a sequence of observations $\mathbf{O} = \{O_1 O_2 \ldots O_T\}$, and a $K \times T$ matrix $\mathbf{B}$, where b_{kt} gives the probability of being in state k given the observation at time t, without any temporal information. These are known as the *emission probabilities.* In a continuous density HMM, the $\mathbf{B}$ matrix is computed as the output of a suitable probability density model given the observation.

The operation of the Viterbi algorithm involves computing the most probable state path to each state at each time, and in calculating the probability of that state path. At each point, we retain a record of which state from the previous time step gave rise to the most probable path. This is illustrated by the trellis diagram in Fig. B.1. The bold arrows represent the most likely path to the given state, from the previous state. Two partial paths through the trellis are shown. The upper path shows that if the state is S_1 at the second time step, then the most likely previous state is S_1. At the third time step, if the state is S_2 then the most likely previous state is also S_1, and the connected path of arrows shows a potential state path.

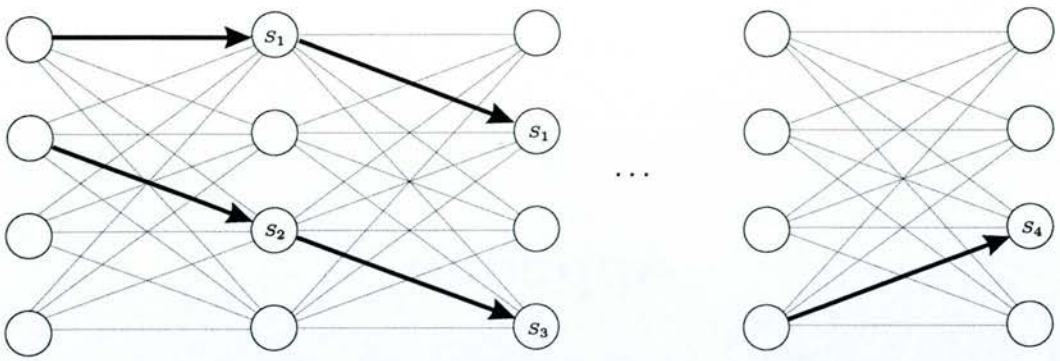


Figure B.1: Trellis diagram illustrating the Viterbi decoding algorithm. Two partial candidate paths are shown, with the bold arrows denoting the most likely previous state leading to this state, given the data. Each node keeps a back-pointer to the most likely previous state. At the end of the sequence, the most likely state at the end is known, and the chain of back-pointers is then used to recover the most likely path for the whole sequence.

Having computed the most likely previous states for all the states, we also need to compute the probabilities of each of the states, which is done using the emission probabilities. These probabilities are then used in the next stage. At the end of the time sequence, we can then find the most likely final state, and are able to backtrack through the trellis, having stored the back pointers to the most likely previous state.

We can now define the algorithm formally, introducing two quantities that need to be computed at each stage. The first is the vector of probabilities at each step, which is the probability of the most likely path through the trellis, given the observations, leading to being in a particular state. We define this probability for step t and state k as δ_{kt}. The inital column of the δ is set up as the emission probability of the first observation multiplied by the prior. We also need to keep a record of the previous state number that gave rise to this probability (the numbers stored in the circular nodes in Fig. B.1). We shall define ψ_{it} as the most likely state at time $t-1$ leading to state S_i at time t. The state at time t is defined as q_t. The pseudo-code for the algorithm is listed as Algorithm 1.

Algorithm 1 Viterbi Algorithm. Computes most probable state sequence $\{q_t^{\text{best}}; t = 1, \ldots, T\}$ given model $\lambda = \{\mathbf{A}, \boldsymbol{\pi}\}$ and the matrix $\mathbf{B}$ of emission probabilities of the observation sequence.

{*Initialization*}
for $k = 1, 2, \ldots, K$ **do**
  $\delta_{k1} = \boldsymbol{\pi}_k b_{k1}$
end for
{*Recursion through time*}
for $t = 2, \ldots, T$ **do**
  for $j = 1, \ldots N$ **do**
    $\delta_{jt} = \left(\max_{1 \le k \le K} (a_{ij} \delta_{t-1,i}) \right) b_{jt}$
    $\psi_{jt} = \underset{1 \le k \le K}{\operatorname{argmax}} (a_{ij} \delta_{t-1,i})$
  end for
end for
{*Compute end state of most likely path*}
$q_T^{\text{best}} = \underset{1 \le k \le K}{\operatorname{argmax}} [\delta_{kT}]$
{*Backtrack through trellis*}
for $t = \{T-1, T-2, \ldots, 1\}$ **do**
  $q_t^{\text{best}} = \psi_{t+1, q_{t+1}^{\text{best}}}$
end for

Appendix C

Jensen's Inequality applied to the EM algorithm

The key step in the EM algorithm is the appeal to Jensen's inequality, that allows us to form a lower bound on the likelihood function of $P(\mathbf{x})$ in terms of an arbitrary distribution $Q(\mathbf{x})$. The lower bound is a maximum if $Q(\mathbf{x}) = P(\mathbf{x})$. Jensen's inequality is proven in terms of the properties of convex functions.

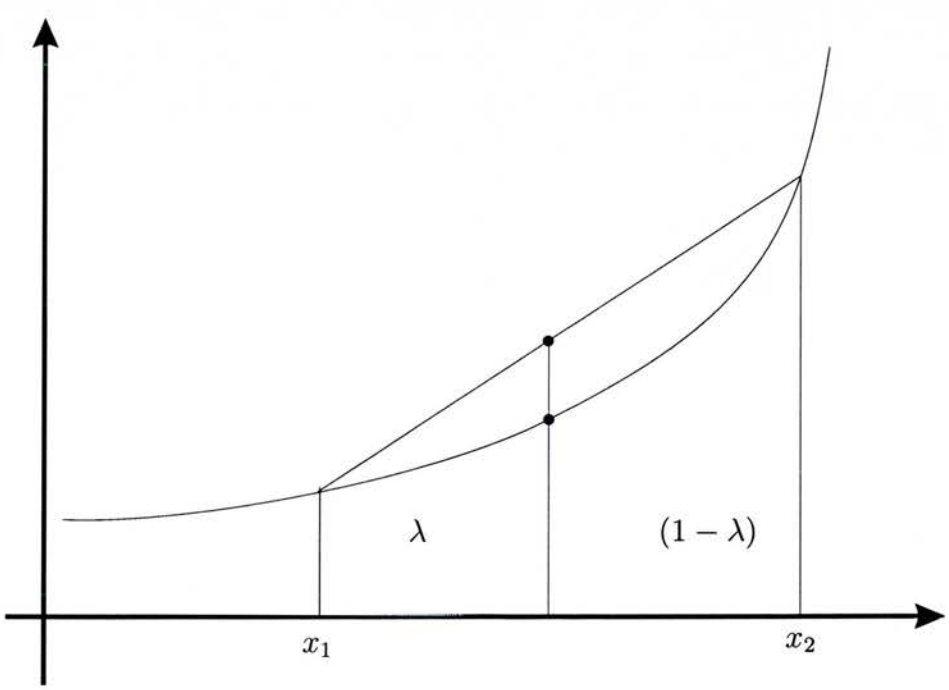


Figure C.1: Convex function.

A function is convex on the interval (a, b) if for every pair (x_1, x_2) on the interval, and for $0 \leq \lambda \leq 1$, the following inequality holds:

$$f(\lambda x_1 + (1-\lambda)x_2) < \lambda f(x_1) + (1-\lambda) f(x_2) \tag{C.1}$$

This can be understood from Fig. C.1, which shows that the line segment always lies above the curve.

Jensen's inequality applied to probability distributions implies that $Ef(\mathbf{x}) \geq f(E\mathbf{x})$ for a convex function $f(.)$, with the inequality the other way round for a concave function. So, we have:

$$\sum_{\mathbf{x}} p(\mathbf{x}) f(\mathbf{x}) \geq f\left(\sum_{\mathbf{x}} p(x) x\right) \tag{C.2}$$

The proof of the inequality follows from induction. We know that it is true for a summation of two terms from the definition of complexity, because

$$p_1 f(\mathbf{x}_1) + p_2 f(\mathbf{x}_2) \geq f(p_1 \mathbf{x}_1 + p_2 \mathbf{x}_2), \tag{C.3}$$

and $p_2 = 1 - p_1$. For the inductive step, we assume that it works for $k-1$ points, and we introduce the term $\tilde{p}_i = p_1/(1-p_k)$. Then:

$$\begin{aligned}
\sum_{i=1}^{k} p_i f(\mathbf{x}_i) &= p_k f(\mathbf{x}_k) + (1-p_k) \sum_{i=1}^{k-1} \tilde{p}_i f(\mathbf{x}_i) && \text{(C.4)} \\
&\geq p_k f(\mathbf{x}_k) + (1-p_k) f\left(\sum_{i=1}^{k-1} \tilde{p}_i \mathbf{x}_i\right) && \text{(C.5)} \\
&\geq f\left(p_k \mathbf{x}_k + (1-p_k) \sum_{i=1}^{k-1} \tilde{p}_i \mathbf{x}_i\right) && \text{(C.6)} \\
&= f\left(\sum_{i=1}^{k} p_i \mathbf{x}_i\right) \tilde{p}_i f(\mathbf{x}_i). && \text{(C.7)}
\end{aligned}$$

The first inequality in the above is the inductive assumption that it is true for $k-1$ points, and the second arising from the definition of a convex function.

At the heart of the E-M algorithm, we establish a lower bound on the log likelihood function, noting that:

$$\int_{\mathbf{x}} Q(\mathbf{x}) \log \frac{Q(\mathbf{x})}{P(\mathbf{x})} d\mathbf{x} \leq \log \int_{\mathbf{x}} Q(\mathbf{x}) \frac{P(\mathbf{x})}{Q(\mathbf{x})} d\mathbf{x} \tag{C.8}$$

$$= \log \int_{\mathbf{x}} P(\mathbf{x}) d\mathbf{x} = 0 \tag{C.9}$$

So the maximum of the function has the value zero, and occurs only when $P(\mathbf{x}) = Q(\mathbf{x})$. Hence the maximization of the E step is achieved by setting the distribution to the posterior distribution of the latent variables given the current model parameters. In practice, we calculate sufficient statistics for the M-step; the expected values of the distribution parameters. Hence the term "Expectation".

Bibliography

Bell, A. and Sejnowski, T. (1996). Learning the higher-order structure of a natural sound. *Network: Computation in Neural Systems*, 7:261–266.

Bell, A. J. and Sejnowski, T. J. (1995). An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1129–1159.

Bellman, R. (1961). *Adaptive Control Processes: A Guided Tour*. Princeton University Press, New Jersey.

Bishop, C. (1995). *Neural Networks for Pattern Recognition*. OUP, Oxford, U.K.

Bishop, C., Strachan, I., O'Rourke, J., Maddison, G., and Thomas, P. (1993). Reconstruction of tokamak density profiles using feedforward networks. *Neural Computing & Applications*, 1(1):4–16.

Bishop, C. M. (1994). Mixture density networks. Technical report, Neural Computing Research Group, Aston University, Birmingham, B4 7ET, U.K.

Bishop, C. M., Hinton, G. E., and Strachan, I. G. D. (1997a). GTM through time. In *Proceedings IEE Fifth International Conference on Artificial Neural Networks*, pages 111–116, London. IEE.

Bishop, C. M., Svensén, M., and Williams, C. K. I. (1997b). Magnification factors for the GTM algorithm. In *Proceedings IEE Fifth International Conference on Artificial Neural Networks*, pages 64–69, London. IEE.

Bishop, C. M., Svensén, M., and Williams, C. K. I. (1997c). Magnification factors for the SOM and GTM algorithms. In *Proceedings 1997 Workshop on Self-Organizing Maps, Helsinki, Finland*.

Bishop, C. M., Svensén, M., and Williams, C. K. I. (1998a). Developments of the Generative Topographic Mapping. *Neurocomputing*, 21:203–224.

Bishop, C. M., Svensén, M., and Williams, C. K. I. (1998b). GTM: The generative topographic mapping. *Neural Computation*, 10(1):215–234.

Bishop, C. M. and Tipping, M. E. (1996). A hierarchical latent variable model for data visualization. Technical report, Neural Computing Research Group, Aston University, Birmingham B4 7ET, U.K.

Broomhead, D. and Lowe, D. (1988). Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2(3):321–355.

Chappelier, J.-C. and Grumbach, A. (1995). A Kohonen map for temporal sequences. In *Proceedings of NEURAP'95, Marseille, France*, pages 104–110.

Chappell, G. and Taylor, J. (1993). The Temporal Kohonen Map. *Neural Networks*, 6:441–445.

Corti, S. (1980). The problem of abel inversion in the determination of plasma density from phase measurements. *Let. Nuovo Com.*, 29:25–32.

Davis, S. and Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Trans Acoust, Speech, and Signal Process*, 28(4):357–366.

Dempster, A. P., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, B*, 39:1–38.

Dongarra, J., Moler, C., Bunch, J., and Stewart, G. (1979). *Linpack User's Guide*. SIAM, Philadelphia.

Ford, J. and Moore, J. (1998). Adaptive estimation of hmm transition probabilities. *IEEE Transactions on Signal Processing*, 46(5):1374–1385.

Forney, G. (1973). The Viterbi Algorithm. *Proc. IEEE*, 61:268–278.

Friedman, J. and Tukey, J. (1974). A projection pursuit algorithm for exploratory data analysis. *IEEE Trans. Comput.*, C-23:881–889.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58.

Ghahramani, Z. and Hinton, G. (1996a). Parameter estimation for linear dynamical systems. Technical Report CTG-TR-96-2, University of Toronto.

Ghahramani, Z. and Hinton, G. (1996b). Switching state-space models. Technical Report CTG-TR-96-3, University of Toronto.

Haese, K. and Goodhill, G. (2001). Auto-SOM: Recursive parameter estimation for guidance of self-organizing feature maps. *Neural Computation*, 13:595–619.

Hecht-Nielsen, R. (1992). The munificence of high dimensionality. In Aleksander and Taylor, editors, *Artificial Neural Networks, 2*, pages 1017–1030. North-Holland.

Honkela, T., Pulkki, V., and Kohonen, T. (1995). Contextual relations of words in grimm tales, analyzed by self organizing map. In Fogelman-Soulie, F. and Gallinari, P., editors, *Proceedings of International Conference on Artificial Neural Networks, ICANN-95*, pages 3–7. EC2 et Cie, Paris.

Hyvarinen, A. (1999). Survey on independent component analysis. *Neural Computing Surveys*, 2:94–128.

Hyvirinen, A. and Oja, E. (1997). A fast fixed-point algorithm for independent component analysis. *Neural Computation*, 9(7):1483–1492.

Jollife, I. (1986). *Principal Component Analysis.* Springer-Verlag, New York.

Kaban, A. and Girolami, M. (2002). A dynamic probabilistic model to visualise topic evolution in text streams. *Journal of Intelligent Information Systems*, 18. To appear (current version obtainable from http://cis.paisley.ac.uk/kaba-ci0/).

Kohonen, T. (1989). *Self-Organization and Associative Memory.* Springer, Berlin, third edition.

Kohonen, T. (1995). *Self-Organizing Maps.* Springer, Berlin.

Kohonen, T., Kaski, S., Lagus, K., and Honkela, T. (1996). Very large two-level SOM for the browsing of newsgroups. In von der Malsburg, C., von Seelen, W., Vorbrüggen, J. C., and Sendhoff, B., editors, *Proceedings of ICANN96, International Conference on Artificial Neural Networks, Bochum, Germany, July 16-19, 1996*, Lecture Notes in Computer Science, vol. 1112, pages 269–274. Springer, Berlin.

Kohonen, T., Kaski, S., Lagus, K. K., Salojaervi, J., Paatero, V., and Saarela, A. (2000). Organization of a massive document collection. *IEEE-NN*, 11(3):574.

Kurimo, M. (1997). Training mixture density HMMs with SOM and LVQ. *Computer Speech and Language*, 11(4):321–343.

Lowe, D. (1998). Feature space embeddings for extracting structure from single channel wake eeg using rbf networks. In *In Neural Networks for Signal Processing:Proceedings of the IEEE workshop*, pages 428–437.

Lowe, D. and Tipping, M. E. (1996). Feed-forward neural networks and topographic mappings for exploratory data analysis. *Neural Computing and Applications*, 4:83–95.

MacKay, D. (1992). A practical bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472.

Makhoul, J. (1975). Linear prediction: A tutorial review. *Proceedings of the IEEE*, 63(4):561–580.

Mulier, F. and Cherkassky, V. (1995). Self-organization as an iterative kernel smoothing process. *Neural Computation*, 7(6):1165–1177.

Neal, R. M. and Hinton, G. E. (1998). A new view of the EM algorithm that justifies incremental, sparse and other variants. In Jordan, M. I., editor, *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers.

Oja, E., Ogawa, H., and Wangviwattana, J. (1992). Pca in fully parallel neural neworks. In Aleksander and Taylor, editors, *Artificial Neural Networks, 2*, pages 199–203. North-Holland.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems.* Morgan–Kaufman, San Mateo.

Pearlmutter, B. A. and Parra, L. C. (1997). Maximum likelihood blind source separation: A context-sensitive generalization of ICA. In Mozer, M. C., Jordan, M. I., and Petsche, T., editors, *Advances in Neural Information Processing Systems*, volume 9, page 613. The MIT Press.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proc. IEEE*, 77(2).

Rauch, H., Tung, F., and Striebel, C. (1965). Maximum likelihood estimates of linear dynamic systems. *AIAA J.*, 3(8):1445–1450.

Reinhard, K. and Niranjan, M. (1998). Parametric subspace modeling of speech transitions. Technical Report CUED/F-INFENG/TR.308, Cambridge University Engineering Department.

Reynolds, J. and Tarassenko, L. (1993). Learning pronunciation with the visual ear. *Neural Computing & Applications*, 1(3):169–175.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407.

Roweis, S. (1999). Constrained Hidden Markov Models. In *NIPS 12*, pages 782–788, Cambridge,MA. MIT Press.

Roweis, S. and Ghahramani, Z. (1999). A unifying review of linear gaussian models. *Neural Computation*, 11(2):305–345.

Roweis, S. and Ghahramani, Z. (2000). An EM algorithm for identification of nonlinear dynamical systems. *Preprint under revision for IEEE Transactions on Automatic Control.* Available at `http://www.gatsby.ucl.ac.uk/~zoubin/papers/nlds_preprint.ps.gz`.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back–propagating errors. *Nature*, 323:533–536.

Sadie, S., editor (2001). *The New Grove Dictionary of Music and Musicians.* MacMillan.

Sammon, J. (1969). A nonlinear mapping for data analysis. *IEEE Transactions on Computers C-18*, 5:401–409.

Stiller, J. and Radons, G. (1999). Online Estimation of Hidden Markov Models. *IEEE Signal Processing Letters*, 6(8):213–215.

Svensén, M. (1998). *GTM: The Generative Topographic Mapping.* PhD thesis, Aston University, Birmingham, U.K.

Tarassenko, L., Townsend, N., Clifford, G., Mason, L., Burton, J., and Price, J. (2001). Medical signal processing using the software monitor. In *Proceedings DERA/IEE Workshop on Intelligent Signal Processing*, pages 3/1–3/4, Birmingham.

Tipping, M. E. (1996). *Topographic Mappings and Feed-forward Neural Networks.* PhD thesis, Aston University.

Tipping, M. E. and Bishop, C. M. (1997). Probabilistic Principal Component Analysis. Technical report, Neural Computing Research Group, Aston University, Aston Triangle, Birmingham, B4 7ET, U.K.

Tipping, M. E. and Bishop, C. M. (1999). Mixtures of probabilistic principal component analyzers. *Neural Computation*, 11(2):443–482.

Tipping, M. E. and Lowe, D. (1998). Shadow targets: a novel algorithm for topographic projections by radial basis functions. *Neurocomputing*, 19(1):211–222.

Utete, S., Tarassenko, L., Hayton, P., Hesketh, G., and Anuzis, P. (2000). Data fusion for jet engine condition monitoring. In *Proceedings of NCAF Summer Meeting*, Manchester.

Varsta, M., Heikkonen, J., and del R. Millan, J. (1997). Context learning with the self organizing map. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4–6*, pages 197–202. Helsinki University of Technology, Neural Networks Research Centre, Espoo, Finland.

Varsta, M., Heikkonen, J., and Lampinen, J. (2000). Analytical comparison of the Temporal Kohonen Map and the Recurrent Self Organizing Map. In Verleysen, M., editor, *Proc. ESANN'2000*, pages 273–280, Bruges, Belgium. D-Facto.

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Trans. Informat. Theory*, IT-13:260–269.